Proyecto de Sistemas Informáticos

Curso Académico 2007 / 2008

Facultad de Informática

Universidad Complutense de Madrid

DataBase Design Tool

"Una herramienta para el diseño de Bases de Datos."

Autores *Alberto Milán Gutiérrez*

Miguel Martínez Segura

Fco. Javier Cáceres González

Profesor director *Yolanda García Ruiz*

Autorizaciones de uso.

Se autoriza a la Universidad Complutense difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Firma de los autores:

Alberto Milán Gutiérrez

Miguel Martínez Segura

Fco. Javier Cáceres González

Agradecimientos.

Este proyecto no hubiese sido posible de llevar a cabo si no es por la colaboración de personas a las que nos gustaría agradecer su confianza y ayuda.

A Yolanda García Ruiz, por aceptar dirigirnos el proyecto y permitirnos actuar con total libertad creativa asesorándonos para muchas de las situaciones complejas en las que nos hemos encontrado.

A Rafael Caballero Roldán, por confiar en nuestra propuesta y ponernos en contacto con Yolanda, amen de las pruebas que realizó de nuestra aplicación.

A nuestras novias, familiares y amigos, por entender que estar delante de un ordenador a altas horas de la madrugada ya no significa que se esté jugando a algo.

Al personal de la cafetería, concretamente a Manolo y Sánchez, que con sus cafés, cervezas y bocadillos han duplicado la productividad de los desarrolladores y conservado su estado de ánimo en los momentos más delicados.

En fin, muchas gracias a todos.

Introducción.

Español

La aplicación *DataBase Design Tool* es una herramienta para el diseño gráfico de bases de datos relacionales.

El objetivo de la misma es facilitar el diseño de bases de datos relacionales de una forma visual y simple para completar el aprendizaje en asignaturas que lo requieran o realizarlo de forma particular.

Dado que la herramienta está orientada a entornos académicos, tiene una implementación multiplataforma que permite ejecutarla tanto en sistemas *Windows* como en distribuciones *GNU/Linux* (*Debian* y *Ubuntu*).

Al poseer un interfaz gráfico, similar al expuesto en la mayor parte de la bibliografía de uso común, evita la complejidad del desarrollo sobre papel de diagramas que pueden llevar a equívocos o errores, y permite la corrección de dichos esquemas. De la misma forma, el profesor puede desempeñar su función de una forma más cómoda proponiendo soluciones o corrigiendo ejercicios de una manera clara y eficaz.

La aplicación, además de permitir diseñar y comprobar un esquema, realiza la tarea de generar del script en lenguaje SQL para la creación de la base de datos física diseñada. Dicho proceso es automático y simplifica tanto el tiempo como la complejidad de un diseño teórico llevado a la práctica.

El objetivo principal de la herramienta es su uso y desempeño con fines docentes, proporcionando una gran ayuda tanto para enseñar como para aprender la teoría de las bases de datos.

English

The application *DataBase Design Tool* is a tool for graphic design of relational databases.

Its purpose is to facilitate the design of relational databases in a simple and visual form to complete the learning in subjects that require it or to do it on your own.

Having in mind that the tool is aimed to an academic environment, it has a multi-platform implementation, that make possible to use it whether in *Windows* systems or in *GNU/Linux* distributions (*Debian* and *Ubuntu*)

By having a graphical interface, similar to the one described in the most commonly used literature, it avoids the complexity of development on paper charts that can lead to misunderstandings or mistakes, and allows the verification of the correctness of such schemes. In the same way, the teacher can perform his function in a more comfortable way proposing solutions or correcting exercises in a clear and effective way.

The application, in addition to make possible to design and test a scheme, performs the task of generating a SQL language script for the creation of the physical database designed. This process is automatic and simplifies both time and complexity of a theoretical design implemented in practice.

The main purpose of the tool is its use and performance for teaching purposes, in order to help either the teaching or the learning of databases theory.

Objetivos del proyecto.

Cuando nos planteamos la idea de realizar el proyecto de Sistemas Informáticos, pensamos en nuestras propias experiencias vividas como alumnos. La idea de facilitar el acercamiento al diseño de Bases de Datos Relacionales ha sido el eje central para el desarrollo de esta aplicación.

Los objetivos de este proyecto han estado desde el primer momento muy claros. La idea principal ha sido crear una herramienta con fines docentes que facilitase, tanto al alumno como al profesor la enseñanza de las Bases de Datos Relacionales.

Para poder llevar a cabo esta idea, la primera premisa era conseguir una interfaz amigable y cómoda para poder crear un diagrama Entidad Relación fácilmente. Estos diagramas abarcan el uso de la mayoría de representaciones conceptuales que se requieren para poder crear una Base de Datos bien estructurada.

El segundo punto esencial en los objetivos de la aplicación era la validación de los diseños del usuario. En función de los criterios de corrección, es posible detectar fallos en el diseño de los modelos Entidad Relación. Esto facilita una mejor comprensión por parte del alumno de las directrices que debe tomar al crear una Base de Datos Relacional.

El último gran objetivo que tiene el proyecto es la Generación de Código SQL (Estándar ANSI – ISO 9075) para la creación de Bases de Datos a partir de los diseños del usuario. La aplicación es capaz de interpretar el diagrama Entidad Relación una vez validado y proporcionar un *script* con las sentencias necesarias para construir una estructura física de tablas.

Palabras clave.

Listado de palabras clave usadas en DBDT:

- Bases de Datos Relacionales
- Diagrama Entidad Relación
- Validación.
- Modelo Relacional
- SQL
- Controlador
- Presentación
- Servicios de aplicación
- Persistencia
- XML

Índice de contenidos.

Autorización de uso 3
Agradecimientos 4
Introducción 5
Objetivos del proyecto 6
Palabras clave 7
Índice de contenidos 8

Teoría de las Bases de Datos 12
1. Conceptos básicos 12
2. Modelo Entidad Relación 12
    1. Representación de objetos mediante entidades 12
    2. Representación de atributos 13
    3. Representación de relaciones 13
    4. Representación de restricciones estáticas 13
    5. Representación gráfica 14
3. Modelo Relacional 15
4. SQL (Structured Query Lenguaje) 16
    1. Dominios 16
    2. Lenguaje de definición de datos (DDL) 17
5. Traducción a tablas 19
6. Conceptos cubiertos por la aplicación 20
7. Posibles ampliaciones 21

Especificación de requisitos
1. Introducción 23
2. Requisitos Software específicos 24
    1. Módulo Entidades 24
    2. Módulo Atributos 29
    3. Módulo Relaciones 35
        1. Relaciones normales 35
        2. Relaciones de herencia IsA 41
    4. Módulo Sistema 46

3. Diagramas de secuencia 49
    1. Insertar una entidad 49
    2. Editar la cardinalidad de una entidad en una relación 59
    3. Eliminar un atributo de una entidad 51
    4. Generar el Modelo Relacional 52

Arquitectura del sistema

1. Arquitectura Multicapa 54
2. Patrones de diseño
    1. Modelo Vista Controlador (MVC) 55
    2. Patrón Transferencia (Transfer) 56
    3. Patrón Objeto de Acceso a Datos (DAO) 57

Implementación de sistema

1. Introducción 59
2. Organización de paquetes 60
3. Presentación 61
    1. Interfaces de acción/decisión 62
    2. Diálogo de cambio de espacio de trabajo (Workspace) 63
    3. Frame Principal 64
    4. Marcos de diseño 66
        1. El Panel de Diseño 66
        2. El Panel de Pre-Visualización (o thumbnail) 67
        3. El Panel de Información 68
4. Controlador 69
5. Servicios de aplicación 70
    1. Servicios de Entidades 71
    2. Servicios de Atributos 72
    3. Servicios de Relaciones 73
    4. Servicios de Sistema 74
        1. Validación del diseño 74
            1. Validación de atributos 75
            2. Validación de entidades 76
            3. Validación de relaciones 76

4. Validación de relaciones de herencia 77
5. Validación de relaciones normales 77
6. Validación de relaciones débiles 77

2. Generación de código SQL y del Modelo Relacional 77
1. La clase Tabla 78

3. Implementación específica 78

6. Persistencia 80
1. XML (Extensible Markup Language) 80
2. Ficheros XML en DBDT 80
1. Entidades.xml 80
2. Atributos.xml 81
3. Relaciones.xml 82
3. Implementación de los DAOs 82
4. La clase EntidadYAridad.java 83
7. Librerías externas utilizadas 84

Manual de usuario

1. Introducción 87
2. Requisitos del sistema 87
3. Instalación y ejecución de la aplicación 87
4. Glosario 88
5. Empezando a usar DBDT 90
1. Crear un nuevo proyecto 90
2. Abrir un proyecto existente 92
3. Añadir entidades, atributos y relaciones a un proyecto 93
4. Editar entidades, atributos y relaciones de un proyecto 96
5. Validar el diseño del diagrama E/R 99
6. Generar el Modelo Relacional 99
7. Generación del código SQL 100
6. Ejemplo completo de uso: Base de Datos para un colegio 101

Conclusiones y trabajo futuro 106
Referencias 107

Teoría de las Bases de Datos.

El diseño de una Base de Datos es una de las tareas más importantes en el desarrollo de cualquier aplicación que necesite manejar un conjunto de datos de forma eficiente y organizada.

En este capítulo se estudian brevemente dos herramientas que permiten organizar las propiedades *estáticas* (entidades, atributos y relaciones entre entidades) y *dinámicas* (operaciones entre entidades o sus relaciones) de una parcela del mundo real que es caso de estudio; se trata del Modelo Relacional y el Modelo Entidad Relación.

El objetivo de esta dosis de teoría no es otro que el acercar al lector a los conceptos que abarca la aplicación y por lo tanto a la capacidad operativa de ésta.

Cabe destacar que todas las ilustraciones que aparecen en esta sección, están sacadas de la propia aplicación DBDT.

1. Conceptos básicos.

Entidad: Menor objeto con significado de una instancia, distinguible en el mundo real de todos los demás objetos.

Relación: Asociación entre diferentes entidades.

Atributos: Propiedades que caracterizan a cada conjunto de entidades o relaciones.

Restricciones de integridad:

- Restricciones sobre atributos (*dominio*): conjunto de valores permitidos que puede tomar un determinado atributo.
- Restricciones sobre objetos (definición de claves): limitan el conjunto de ocurrencias posibles de un objeto.
- Restricciones sobre relaciones (*cardinalidad* y *dependencia*).

2. Modelo Entidad Relación.

El modelo Entidad Relación permite definir el esquema conceptual de una Base de Datos. La colección de objetos básicos que representan la información del sistema son las entidades, los atributos y las relaciones. Cada componente tiene una representación gráfica que mostraremos mas adelante.

Este modelo proporciona un lenguaje de definición de tipo gráfico para representar cada uno de los conceptos citados anteriormente.

2.1. Representación de objetos mediante entidades.

En los diagramas se representan lo que se denominan *conjuntos de entidades* con la finalidad de englobar todas las entidades de un mismo tipo. Por ejemplo, los alumnos de una universidad serian un conjunto de entidades representadas como *alumno* dentro del diagrama.

Los conjuntos de entidades no son disjuntos. Dado que representan objetos del mundo real, las limitaciones de estos conjuntos están sujetas la mayoría de las veces a los criterios

subjetivos del diseñador. Así, *alumno* puede ser un subconjunto de un conjunto de entidades *persona*, tal y como lo seria *profesor*.

2.2. Representación de atributos.

Cada entidad se representa mediante un conjunto de atributos. Estos atributos describen las propiedades que posee cada miembro de un conjunto de entidades.

Existen dos variantes claras dentro de las características de un atributo.

- *Simples* o *compuestos*: Cuando un atributo puede estar dividido en subpartes, se le denomina compuesto. Un ejemplo claro es una dirección postal. Constara del nombre de la calle, del número y del piso. Evidentemente, cuando es simple no tiene subpartes.
- *Multivalorados* y *monovalorados*: Cuando un atributo puede tomar más de un valor para una entidad determinada se le considera multivalorado. Un ejemplo podría ser un número de teléfono. Este tipo de atributos no pueden ser clave de ningún conjunto de entidades.

2.3. Representación de relaciones.

Una relación es una asociación entre diferentes entidades. Pueden tener atributos descriptivos, por ejemplo, en una relación *imparte* en la que intervengan las entidades *profesor* y *asignatura*, puede haber un atributo *horario*.

Las asociaciones pueden darse entre dos o más entidades, denominándose relaciones binarias, ternarias, etc. Esto es lo que se conoce como *grado* de la relación.

2.4. Representación de restricciones estáticas.

Restricciones sobre atributos.

El conjunto de valores permitidos que puede tomar un determinado atributo se denomina *dominio*.

Restricciones sobre objetos (entidades).

Cuando un atributo o un conjunto de éstos identifican inequívocamente a una entidad, se le considera *clave* de esa entidad. Bien es verdad que este tipo de decisiones dependen del criterio del diseñador, ya que inicialmente se consideran las *superclaves* y las *claves candidatas* y de ahí se deducen las *claves primarias*.

Restricciones sobre relaciones.

Las restricciones sirven para definir las limitaciones de las entidades que intervengan en una relación y por lo tanto de todas las del diagrama Entidad Relación.

Las limitaciones a las que nos referimos están centradas en la razón de *cardinalidad* de las entidades. Cuando una entidad se relaciona a lo sumo con una entidad de otro conjunto se considera relación *uno a uno*. Cuando tenemos el caso en el que la entidad se puede asociar con cualquier número de entidades del otro conjunto se habla de relaciones varios a varios. Una situación equivalente se da para *uno a varios* y *varios a uno*. Cabe destacar que es posible limitar el numero de entidades de forma concreta es decir, se pueden relacionar de una a ocho (1..8) entidades con de una a seis (1..6) entidades de otro conjunto. La aplicación DBDT esta basada en este concepto.

Restricciones de dependencia en identificación.

Un conjunto de entidades que no tenga los suficientes atributos para formar una clave primaria se considera un *conjunto de entidades débiles*. Este tipo de entidades, cuando intervienen en una relación, dependen siempre de otro conjunto de entidades fuertes. No obstante, pese a no poseer atributos clave, si puede tener lo que se conoce como conjunto de atributos discriminantes que tienen una función similar ya que permite distinguir las entidades dentro de un mismo conjunto.

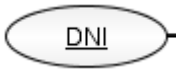
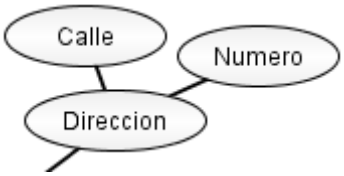
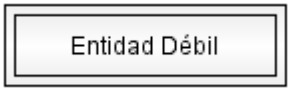
Las relaciones en las que interviene una entidad débil pueden convertirse en lo que se conoce como relaciones débiles.

Representación de generalización o relación de herencia.

Tal como hemos comentado anteriormente, existe la posibilidad de representar conjuntos no disjuntos de entidades que hereden atributos.

2.5. Representación gráfica.

A continuación se mostrará una representación de todos y cada uno de los objetos que pueden formar parte de un diagrama Entidad Relación.

Atributo	Atributo Multivalorado	Atributo Clave Primaria
		
Atributo compuesto	Entidad	Entidad débil
		
Relación normal	Relación con cardinalidad	Relación débil
		
Relación de herencia		
		

3. Modelo Relacional.

El Modelo Relacional es un modelo de datos que ha tomado mucha relevancia como principal modelo de datos para las aplicaciones de procesamiento de éstos. Actualmente es el modelo elegido para casi todos los SGBD comerciales.

Existe una fuerte correspondencia entre el concepto de tabla y el concepto matemático de relación. De ahí el nombre de este modelo. Esta estructura permite además, representar tanto objetos como asociaciones entre éstos.

Una vez obtenidas las tablas, existe lo que se conoce como *álgebra relacional* que permite una serie de operaciones fundamentales sobre los datos de éstas: selección, composición, proyección, unión, diferencia, producto cartesiano, etc.

Esta sección teórica no tiene mucha relevancia en DBDT, ya que esta aplicación se dedica exclusivamente al diseño y no a las operaciones entre tuplas de datos.

A continuación se muestra un ejemplo de un modelo relacional derivado del diseño de una Base de Datos de una Universidad.

Aulas = (Edificio, Numero)

Profesores = (Nombre_y_apellidos, telefono, Domicilio, DNI)

Asignaturas = (Codigo, Titulo, Numero_de_creditos)

Alumnos = (DNI, Nombre_y_apellidos, Domicilio_Calle, Domicilio_Numero, Domicilio_Ciudad, COU)

Imparte = (Nombre_y_apellidos, Codigo, Edificio, Numero)

Matricula = (DNI, Codigo, Nota)

Supervisa = (Nombre_y_apellidos, Nombre_y_apellidos_slave)

Telefono = (DNI, telefono)

4. SQL (Structured Query Lenguaje).

SQL es un lenguaje de consulta estructurado declarativo de alto nivel y de no procedimiento. Actualmente es el lenguaje estándar paro los sistemas de bases de datos relacionales.

Proporciona entre otros componentes un lenguaje de definición de datos *DDL (Data Definition Lenguaje*) y otro de manipulación *DML (Data Manipulation Languaje)*.

En la aplicación DBDT, se usan las sentencias DDL que permiten la creación de una Base de Datos diseñada previamente en un modelo Entidad Relación. El conjunto de sentencias DDL se explicará con más detalle en esta documentación.

Otras posibilidades de SQL son la manipulación de datos, vistas, *triggers*… amen de todas las operaciones fundamentales del álgebra relacional.

A continuación vemos un cuadro resumen de las sentencias más comunes.

Manipulación de datos *(DML)*	**Select * from** Table **where**… **Update** Table **set**… **where**… **Insert into** Table **values**… **Delete from** Table **where**…
Vistas *(Definición)*	**Create view** NombreVista **as**…
Cursores	**Declare** NombreCursor **cursor for…**
Trigger *(PL/SQL)*	**Create Trigger** NombreTrigger **for**…
Algebra relacional	**Inner join**, **outer join**…

4.1. Dominios.

Tal y como se comentaba en la primera parte de esta teoría, cada atributo de una entidad tiene un dominio (valores que puede tomar). En consecuencia toda columna de una tabla también tendrá su dominio y en SQL existen ciertos tipos básicos que han sido implementados en la aplicación. Es importante resaltar que estos dominios no son todos comunes en todos los Sistemas Gestores de Bases de Datos. Por ejemplo, *Oracle* no acepta el tipo *BIT* y *MySQL* si.

Estos son los dominios más habituales:

- *Char*(n): Es una cadena de caracteres con una longitud *n* especificada por el usuario. También se puede usar la palabra completa *character*.
- *VarChar*(n): Es una cadena de caracteres con una longitud variable inicialmente especificada por el usuario.
- *Int*: Es un entero. Se puede usar *Integer*.
- *Real*: Es un real

- *Float*(n): Números en coma flotante de precisión n especificada por el usuario.
- *Date*: Es una fecha del calendario que especifica año, mes y día.
- *Time*: Es la hora del DIA expresada en horas, minutos y segundos.

4.2. Lenguaje de definición de datos (DDL).

Tras el acercamiento al resto de sentencias de SQL presentado anteriormente, nos centramos ahora en el lenguaje DDL, que son el conjunto de sentencias que sirven para definir esquemas de la base de datos. Estas sentencias son las implicadas directamente en la aplicación DBDT y por eso vamos a ser más específicos en esta teoría.

Existen cuatro operaciones básicas: *create*, *alter*, *drop* y *truncate*.

Definición de tablas (Create).

Este comando crea un objeto dentro de la base de datos ya sea una tabla, vista, trigger o cualquier otro soportado por el sistema.

Nuestro uso en la aplicación, se centra en la creación de tablas derivadas de un modelo Entidad Relación previamente diseñado, tal y como mostramos en el ejemplo siguiente.

```
CREATE TABLE Profesores(
        Nombre_y_apellidos CHAR(50),
        teléfono INTEGER,
        Domicilio CHAR(30),
        DNI INTEGER);

CREATE TABLE Asignaturas(
        Codigo INTEGER,
        Titulo CHAR(30),
        Numero_de_creditos INTEGER);

CREATE TABLE Imparte(
        Codigo INTEGER,
        Nombre_y_apellidos CHAR(50),
        Edificio CHAR(20),
        Numero INTEGER);
```

La sentencia esta compuesta por las palabras reservadas *Create Table*, seguidas por el nombre de la tabla en cuestión y sus atributos (acompañados del dominio de cada uno de ellos).

Modificación de tablas (Alter).

Este comando permite modificar la estructura de un objeto. Podemos añadir o quitar columnas a una tabla, modificar dominios de un campo concreto, etc.

En la aplicación DBDT, este comando se utiliza para una de las operaciones mas importantes, establecer las claves de cada tabla ya sean primarias o foráneas. Siguiendo el ejemplo de la anterior sentencia:

```
ALTER TABLE Profesores ADD PRIMARY KEY (Nombre_y_apellidos);
ALTER TABLE Asignaturas ADD PRIMARY KEY (Codigo);
ALTER TABLE Imparte ADD FOREIGN KEY (Nombre_y_apellidos)
                    REFERENCES Profesores (Nombre_y_apellidos);
```

La sentencia esta compuesta por las palabras reservadas *Alter Table*, seguidas del nombre de la tabla en cuestión y la modificación a realizar. En este caso, añadir claves.

Eliminación de tablas (Drop).

Este comando elimina un objeto de la base de datos. Es responsabilidad del diseñador haber incluido cláusulas que conserven la integridad referencial de los datos para a la hora de eliminar una tabla se eliminen la intervención de sus atributos en otras.

```
DROP TABLE Profesores;
DROP TABLE Asignaturas;
DROP TABLE Imparte;
```

La sentencia esta compuesta por las palabras reservadas *Drop Table*, seguidas del nombre de la tabla a eliminar.

Truncamiento de tablas (Truncate).

Este comando trunca todo el contenido de una tabla. Es similar a *Delete,* pero no puede condicionarse con una cláusula *where*. Esto puede resultar una desventaja, pero permite la eliminación de información mucho más rápidamente.

```
TRUNCATE TABLE profesores;
```

Estas son las nociones básicas teóricas sobre las bases de datos relacionales en las que esta basada la aplicación y gracias a las cuales el usuario puede orientarse dentro del funcionamiento de ésta.

5. Traducción a tablas.

La parte mas interesante y estrechamente ligada a la aplicación es el paso de un modelo Entidad Relación a un esquema físico.

A partir de dicho esquema se obtienen tanto el *modelo relacional* anteriormente explicado como el *código SQL* para la creación física de la base de datos diseñada.

Cada conjunto de entidades y cada conjunto de relaciones generará una tabla independiente en el sistema. Evidentemente existen limitaciones para las entidades y relaciones débiles.

Además, se tiene en cuenta la cardinalidad de los papeles de cada entidad en la relación en la que participe.

Es tarea del diseñador que gracias a una buena estructura del diagrama E/R se obtenga un esquema físico óptimo.

Las directrices para la transformación son las siguientes:

- Cada conjunto de entidades fuertes forma una tabla con una columna por atributo incluido el de la clave principal. Cada fila es una entidad de dicho conjunto.
- Cada conjunto de entidades débiles, forma una tabla a partid del conjunto de entidades fuertes del que depende. La clave será el conjunto de atributos clave de dicha entidad fuerte y el atributo discriminante de la débil.
- Cada conjunto de relaciones genera también una tabla. Esta tabla tiene como columnas los atributos descriptivos propios de la relación y las claves primarias de las entidades implicadas.
  Aquí entra en juego el papel de la cardinalidad para establecer las claves de las tablas. Por ejemplo, al tener una relación con una entidad *A* que participa con una cardinalidad (1..n) frente a otra *B* que participa con (1..1), se establece como clave de la tabla el conjunto de atributos clave de A. En los casos más habituales, *varios a varios*, estas tablas tienen como claves los conjuntos de claves de todas las entidades implicadas.
- Las relaciones de herencia tienen un tratamiento especial. Se crean las tablas para todos los elementos de la relación ya sean padres o hijos, pero la clave de cada una de estas tablas será el conjunto de atributos clave del padre.
- Los atributos multivalorados generan una tabla particular cuya clave será el atributo clave de la entidad de la que depende dicho atributo multivalorado.

6. Conceptos cubiertos por la aplicación.

Una vez expuesta la teoría anterior, podemos pasar a enumerar aquellos conceptos que DBDT es capaz de desarrollar e implementar.

- Modelo entidad-relación
  - Representación de objetos (entidades)
  - Representación de atributos
    - Simples y compuestos
    - Monovalorados y multivalorados
  - Representación de relaciones
  - Representación de restricciones estáticas
    - Dominios de atributos
    - Definición de claves primarias
    - Definición de claves foráneas (integridad referencial)
  - Representación de restricciones para las relaciones
    - Cardinalidad.
    - Grados de las relaciones.
  - Representación de restricciones de dependencia en identificación.
    - Entidades débiles
    - Relaciones débiles
  - Representación de generalización o relaciones de herencia
- Modelo Relacional y Código SQL
  - Validación del diseño.
  - Creación de la estructura de tablas.
  - Generación del modelo relacional asociado.
  - Generación de un script SQL para la creación de la base de datos física.

7. Posibles ampliaciones

Planteándonos posibles extensiones del proyecto, llegamos a los siguientes conceptos:

- Definición de atributos derivados.
- Dominios definidos por el usuario con reglas de validación de rangos.
- Especificación de integridad referencial.
- Diferenciación entre jerarquías en especificación e implementación.
- Establecer conexiones con SGBD para el mantenimiento de la BD.

Especificación de requisitos.

1. Introducción.

La descripción de los servicios que ofrece DBDT y las restricciones de los mismos conforman los requisitos del sistema. Mediante el proceso de *Ingeniería de Requisitos* hemos identificado los que tiene nuestra herramienta.

Inicialmente, establecimos los requisitos de usuario y posteriormente los formalizamos como requisitos funcionales:

- Los requisitos de usuario son frases en lenguaje natural que describen lo que debe permitir hacer nuestro sistema y bajo qué condiciones se tienen que producir.
- Los requisitos funcionales determinan los servicios del sistema de forma unívoca y establecen las restricciones de los mismos con todo detalle.

El proceso de Ingeniería de Requisitos es un proceso largo y tedioso: el objetivo principal es determinar qué hay que hacer para cada uno de los requisitos del sistema, sin decir el cómo hay que hacerlo.

Su utilidad se basa en reducir el esfuerzo a la hora de la implementación, proporcionar una guía para la validación y la verificación del sistema y servir como base para mejoras posteriores. Se trata de una especie de contrato en el que se especifica de forma unívoca el comportamiento deseado para cada uno de los requisitos funcionales de nuestro sistema: todo lo que no esté especificado, no se implementa.

Aunque este proceso es muy estricto en cuanto las formas, a la hora de implementar nuestro sistema hemos realizado pequeñas modificaciones en la especificación (olvidos, pequeños errores...) que a grandes rasgos no influyen en el comportamiento inicial deseado.

La Especificación de Requisitos Software (SRS - Software Requirements Specification) es el documento que formaliza todos los requisitos funcionales de un sistema.

En nuestro caso, hemos creado una SRS reducida: seguimos el estándar IEEE Std. 830-1998 [Std_830], pero solamente nos hemos centrado en el apartado 3.2 correspondiente a los Requisitos Funcionales Específicos, ya que es la parte que realmente nos ha resultado útil.

Los requisitos funcionales de DBDT están divididos en cuatro módulos:

1. Módulo Entidades.
2. Módulo Atributos
3. Módulo Relaciones.
4. Módulo Sistema.

Cada módulo agrupa los requisitos relacionados con cada uno de los objetos principales de nuestra aplicación.

En secciones posteriores especificaremos todos y cada uno de los requisitos de nuestro sistema exhaustivamente. Además, con fines ilustrativos, mostraremos el comportamiento deseado para algunos de ellos mediante diagramas de secuencia.

2. Requisitos Software específicos.

Como ya se ha dicho, utilizamos el estándar IEEE Std. 830-1998 para la especificación de los requisitos funcionales específicos de nuestro sistema. De acuerdo a esto, tenemos las siguientes especificaciones para cada uno de los módulos:

2.1. Módulo Entidades.

Los requisitos funcionales correspondientes al módulo de entidades son los siguientes:

1. Insertar una entidad.
2. Renombrar una entidad.
3. Debilitar/fortalecer una entidad.
4. Añadir un nuevo atributo a una entidad.
5. Eliminar una entidad
6. Mover la posición de una entidad.

Función	Insertar una entidad.
Descripción	Añadir una nueva entidad al proyecto en curso.
Entrada	TranferEntidad que contiene el nombre para la nueva entidad que se quiere añadir y la posición en el panel donde se ha pinchado.
Salida	TransferEntidad que contiene el nombre de la nueva entidad, la posición en el panel y el identificador interno que le asigna el sistema.
Origen	GUI_InsertarEntidad.
Destino	Sistema.
Necesita	DAOEntidades.
Acción	Crea una nueva entidad en el proyecto en curso con el nombre que desee el usuario.
Precondición	El nombre para la nueva entidad no puede ser vacío. No puede existir ninguna otra entidad en el proyecto en curso cuyo nombre coincida con el que se desea asignar a la nueva entidad.
Postcondición	En caso de éxito, información sobre la inserción de la nueva entidad. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Renombrar una entidad.
Descripción	Cambiar el nombre a una entidad existente en el proyecto en curso.
Entrada	Vector con 2 elementos: El primero de ellos es un TransferEntidad que contiene toda la información de la entidad que se quiere renombrar. El segundo es una cadena con el nuevo nombre que se pretende asignar a la entidad.
Salida	Vector con 3 elementos: El primero es un TransferEntidad con el nombre de la entidad cambiado El segundo es una cadena con el nuevo nombre que se ha asignado El tercero es una cadena con el antiguo nombre que tenía la entidad.
Origen	GUI_RenombrarEntidad.
Destino	Sistema.
Necesita	DAOEntidades.
Acción	Cambia el nombre a la entidad seleccionada por el usuario en el proyecto en curso con el nuevo nombre que desee el usuario.
Precondición	El nuevo nombre que se desea asignar a la entidad no puede ser vacío. No puede existir ninguna otra entidad en el proyecto en curso cuyo nombre coincida con el que se desea asignar a la entidad a renombrar.
Postcondición	En caso de éxito, información sobre el renombramiento de la entidad. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Debilitar/Fortalecer una entidad.
Descripción	Cambia el carácter de débil a una entidad existente en el proyecto en curso.
Entrada	TransferEntidad que contiene toda la información de la entidad que se quiere debilitar/fortalecer.
Salida	TransferEntidad con la misma información de entrada excepto en el carácter débil que ha sido cambiado.

Origen	GUI_Principal
Destino	Sistema.
Necesita	DAOEntidades.
Acción	Debilita o fortalece la entidad seleccionada por el usuario en el proyecto en curso, pasando a tener el carácter complementario.
Precondición	Ninguna.
Postcondición	En caso de éxito, información sobre la modificación del carácter débil de la entidad. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Añadir un atributo a una entidad
Descripción	Añade un nuevo atributo a una entidad existente en el proyecto en curso.
Entrada	Vector de 2 o 3 elementos: El primero de ellos es un TransferEntidad que contiene toda la información de la entidad a la que se desea añadir el nuevo atributo. El segundo es un TransferAtributo que contiene toda la información introducida por el usuario para ese atributo (nombre, carácter compuesto, carácter multivalorado y dominio). Si el vector tiene un tercer elemento éste es el tamaño del dominio del atributo para aquellos dominios en los que es necesario especificar su tamaño.
Salida	Vector de 2 elementos: El primero de ellos es el TransferEntidad de entrada, con el identificador del nuevo atributo añadido a su lista de atributos. El segundo de ellos es un TransferAtributo igual que el de entrada al que el sistema le ha asignado un identificador único.
Origen	GUI_AnadirAtributoEntidad
Destino	Sistema.
Necesita	DAOEntidades, DAOAtributos.

Acción	Añade a la entidad seleccionada un nuevo atributo con las características que desee para éste el usuario.
Precondición	El nombre que se proporcione para el nuevo atributo no puede ser vacío. Si el dominio del nuevo atributo exige un tamaño, éste debe tener el formato adecuado, esto es, debe ser un valor entero positivo.
Postcondición	En caso de éxito, informar sobre la inserción del nuevo atributo y sobre la modificación de la entidad a la que se le ha insertado. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Eliminar una entidad.
Descripción	Elimina la entidad seleccionada por el usuario del proyecto en curso.
Entrada	TransferEntidad con toda la información de la entidad que se desea eliminar del sistema.
Salida	Vector de 2 elementos: El primero es un TransferEntidad con la entidad que ha sido eliminada. El segundo es un Vector de TransferRelacion: cada uno de estos transfers se corresponden con una relación existente en el sistema que ha sido modificada por la eliminación de la entidad anterior.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAOEntidades, DAOAtributos y DAORelaciones
Acción	Elimina la entidad seleccionada del sistema. Si la entidad interviene en alguna relación, quita las referencias a la entidad en dichas relaciones.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre la eliminación de la entidad y sobre la modificación de las relaciones en las que intervenía. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Si la entidad a eliminar tiene atributos también hay que eliminarlos del sistema.

Función	Mover la posición de una entidad
Descripción	Mueve la entidad seleccionada de una posición a otra en el diagrama E/R.
Entrada	TransferEntidad con toda la información de la entidad a la que se desea cambiar su posición en el diagrama. La nueva posición ya está setteada en el TransferEntidad.
Salida	TransferEntidad a la que se le ha modificado su atributo posición.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAOEntidades
Acción	Cambia el atributo posición de la entidad seleccionada a la nueva posición.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre el cambio de posición de la entidad. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

2.2. Módulo Atributos.

Los requisitos funcionales correspondientes al módulo de atributos son los siguientes:

1. Renombrar un atributo.
2. Editar el dominio de un atributo.
3. Editar el carácter compuesto de un atributo.
4. Añadir un nuevo subatributo.
5. Editar el carácter multivalorado de un atributo.
6. Establecer/quitar un atributo como clave primaria.
7. Eliminar un atributo.
8. Mover la posición de un atributo.

Función	Renombrar un atributo
Descripción	Cambiar el nombre a un atributo
Entrada	Vector con 2 elementos: El primero de ellos es un TransferAtributo que contiene toda la información del atributo que se quiere renombrar. El segundo es una cadena con el nuevo nombre que se quiere asignar al atributo.
Salida	Vector con 2 elementos: El primero es el TransferAtributo de entrada con el nuevo nombre ya asignado. El segundo es una cadena con el antiguo nombre que tenía el atributo.
Origen	GUI_RenombrarAtributo.
Destino	Sistema
Necesita	DAOAtributos.
Acción	Cambia el nombre del atributo seleccionado por el nuevo nombre que desee el usuario.
Precondición	El nuevo nombre que se desee asignar al atributo no puede ser vacío.
Postcondición	En caso de éxito, información sobre el renombrado del atributo. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Editar el dominio de un atributo
Descripción	Cambiar el dominio actual de un atributo.
Entrada	Vector con 2 o 3 elementos: El primero de ellos es un TransferAtributo que contiene toda la información del atributo al que se quiere editar el dominio. El segundo de ellos es una cadena con el nuevo dominio del atributo. Si tiene un tercer elemento, éste es el tamaño del dominio para aquellos dominios en los que es necesario especificar su tamaño.
Salida	TransferAtributo con el nuevo dominio asignado.
Origen	GUI_EditarDominioAtributo.
Destino	Sistema.
Necesita	DAOAtributos.
Acción	Cambia el dominio al atributo seleccionado por el nuevo dominio elegido.
Precondición	Si el dominio del nuevo atributo exige un tamaño, éste debe tener el formato adecuado, esto es, debe ser un valor entero positivo.
Postcondición	En caso de éxito, información sobre el cambio en el dominio del atributo. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Editar el carácter compuesto de un atributo.
Descripción	Cambia el carácter compuesto de un atributo.
Entrada	TransferAtributo que contiene toda la información del atributo al que se quiere cambiar el carácter de compuesto.
Salida	TransferAtributo con el carácter de compuesto modificado.
Origen	GUI_Principal
Destino	Sistema.

Necesita	DAOAtributos.
Acción	Cambia el carácter de compuesto del atributo: Si se trata de un atributo simple, se pone como compuesto y su dominio se pone a nulo. Si es un atributo compuesto, se pone como simple.
Precondición	Ninguna.
Postcondición	En caso de éxito, información sobre el cambio en el carácter de compuesto del atributo. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Si se trata de cambiar un atributo compuesto a simple y el atributo tiene subatributos, éstos se eliminarán también del sistema.

Función	Añadir subatributo.
Descripción	Añade un nuevo subatributo a un atributo.
Entrada	Vector de 2 o 3 elementos: El primero de ellos es un TransferAtributo con toda la información del atributo (padre) al que se le va a añadir un nuevo subatributo. El segundo es un TransferAtributo (hijo) con los datos introducidos por el usuario (nombre, carácter compuesto, carácter multivalorado y dominio). Es el nuevo subatributo. Si el vector tiene un tercer elemento, éste es el tamaño del dominio del subatributo para aquellos dominios en los que es necesario especificar su tamaño
Salida	Vector con 2 elementos: El primero es un TransferAtributo (padre) al que le ha sido modificada su lista de componentes. El segundo es un TransferAtributo (hijo) que ha sido añadido al sistema y establecido como componente del padre.
Origen	GUI_AnadirSubatributoAtributo.
Destino	Sistema.
Necesita	DAOAtributos.
Acción	Añade un nuevo subatributo a un atributo del sistema.
Precondición	El atributo padre tiene que ser un atributo compuesto.

	El nombre del nuevo subatributo no puede ser vacío. Si el dominio del nuevo atributo exige un tamaño, éste debe tener el formato adecuado, esto es, debe ser un valor entero positivo.
Postcondición	En caso de éxito, información sobre la inserción del nuevo atributo y de la modificación del atributo padre. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Editar el carácter multivalorado de un atributo.
Descripción	Cambia el carácter multivalorado de un atributo.
Entrada	TransferAtributo que contiene toda la información del atributo al que se quiere cambiar el carácter de multivalorado.
Salida	TransferAtributo con el carácter de multivalorado modificado.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAOAtributos.
Acción	Cambia el carácter de multivalorado del atributo: Si se trata de un atributo multivalorado, se pone como monovalorado. Si es un atributo monovalorado, se pone como multivalorado.
Precondición	Ninguna.
Postcondición	En caso de éxito, información sobre el cambio en el carácter de multivalorado del atributo. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Establecer/quitar atributo como clave primaria.
Descripción	Establece o quita el atributo seleccionado como clave primaria de la entidad a la que pertenece.

Entrada	Vector de 2 elementos: El primero de ellos es un TransferEntidad con toda la información de la entidad a la que pertenece el atributo. El segundo es un TransferAtributo con el atributo que se quiere establecer como clave primaria.
Salida	Vector de 2 elementos: El primero es un TransferEntidad con la entidad a la que se ha añadido/quitado el atributo de su lista de claves primarias. El segundo es un TransferAtributo al que se le ha cambiado el atributo de si es o no clave primaria.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAOEntidades.
Acción	Añade o quita de la lista de claves primarias de la entidad el atributo que ha sido seleccionado.
Precondición	El atributo que se pretende establecer/quitar como clave primaria debe ser un atributo directo de una entidad, es decir, no puede ser un atributo de una relación o subatributo de otro atributo.
Postcondición	En caso de éxito, información sobre el cambio realizado en la lista de claves primarias de la entidad. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Ninguno.

Función	Eliminar un atributo
Descripción	Elimina del sistema el atributo seleccionado.
Entrada	TransferAtributo con toda la información del atributo que se desea eliminar del sistema.
Salida	Vector de 2 elementos: El primero es un TransferAtributo con el atributo que ha sido eliminado del sistema. El segundo elemento es un Transfer que contiene el elemento que ha sido modificado al eliminar el atributo, quitándole de su lista de atributos: este Transfer puede ser un TransferEntidad, un

	TransferRelacion o un TransferAtributo dependiendo del elemento que lo contenga en su lista de atributos (entidades y relaciones) o en su lista de componentes (atributos compuestos).
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAOEntidades, DAOAtributos y DAORelaciones
Acción	Elimina el atributo seleccionado del sistema. Elimina la referencia al atributo del elemento que lo contenga.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre la eliminación del atributo y sobre la modificación del elemento que lo contenía. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Si se trata de un atributo compuesto, hay que eliminar también todos sus subatributos.

Función	Mover la posición de un atributo
Descripción	Mueve el atributo de una posición a otra en el diagrama E/R.
Entrada	TransferAtributo con toda la información del atributo al que se desea cambiar su posición en el diagrama. La nueva posición ya está setteada en el TransferAtributo.
Salida	TransferAtributo al que se le ha modificado su atributo posición.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAOAtributos.
Acción	Cambia el atributo posición del atributo a la nueva posición.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre el cambio de posición del atributo. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Ninguno.

2.3. Módulo Relaciones.

DBDT trabaja con dos tipos de relaciones: relaciones normales y relaciones de herencia (IsA). A las relaciones normales las denominaremos solamente como relaciones, mientras que a las relaciones de herencia las llamaremos relaciones IsA.

Por ello expondremos los requisitos funcionales de acuerdo a esta clasificación.

2.3.1. Relaciones normales.

Los requisitos funcionales correspondientes a esta parte son los siguientes:

1. Insertar una relación.
2. Renombrar una relación.
3. Debilitar/fortalecer una relación.
4. Añadir un atributo a una relación.
5. Añadir una entidad a una relación.
6. Quitar una entidad de una relación.
7. Editar la cardinalidad de una entidad en una relación.
8. Eliminar una relación.

Función	Insertar una relación.
Descripción	Añadir una nueva relación al proyecto en curso.
Entrada	TranferRelacion que contiene el nombre para la nueva relación que se quiere añadir y la posición en el panel donde se ha pinchado.
Salida	TransferRelacion que contiene el nombre de la nueva relación, la posición en el panel y el identificador interno que le asigna el sistema.
Origen	GUI_InsertarRelacion.
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Crea una nueva relación en el proyecto en curso con el nombre que desee el usuario.
Precondición	El nombre para la nueva relación no puede ser vacío. No puede existir ninguna otra relación en el proyecto en curso cuyo nombre coincida con el que se desea asignar a la nueva relación.
Postcondición	En caso de éxito, información sobre la inserción de la nueva relación. Si no se ha podido realizar, información explicativa sobre la razón.

Efectos laterales	Ninguno.

Función	Renombrar una relación.
Descripción	Cambiar el nombre a una relación existente en el proyecto en curso.
Entrada	Vector con 2 elementos: El primero de ellos es un TransferRelacion que contiene toda la información de la relación que se quiere renombrar. El segundo es una cadena con el nuevo nombre que se pretende asignar a la relación.
Salida	Vector con 3 elementos: El primero es un TransferRelacion con el nombre de la relación cambiado. El segundo es una cadena con el nuevo nombre que se ha asignado. El tercero es una cadena con el antiguo nombre que tenía la relación.
Origen	GUI_RenombrarRelacion.
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Cambia el nombre a la relación seleccionada en el proyecto en curso con el nuevo nombre que desee el usuario.
Precondición	La relación a renombrar no puede ser una relación IsA. El nuevo nombre que se desea asignar a la relación no puede ser vacío. No puede existir ninguna otra relación en el proyecto en curso cuyo nombre coincida con el que se desea asignar a la relación a renombrar.
Postcondición	En caso de éxito, información sobre el renombramiento de la relación. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Debilitar/Fortalecer una relación.
Descripción	Cambia el carácter de débil a una relación existente.

Entrada	TransferRelacion que contiene toda la información de la relación que se quiere debilitar/fortalecer.
Salida	TransferRelacion con la misma información de entrada excepto en el carácter débil que ha sido cambiado (si era débil, pasa a ser fuerte y viceversa).
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Debilita o fortalece la relación seleccionada por el usuario en el proyecto en curso, pasando a tener el carácter complementario.
Precondición	La relación a debilitar/fortalecer no puede ser de tipo IsA.
Postcondición	En caso de éxito, información sobre la modificación del carácter débil de la relación. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Si se trata de debilitar una relación fuerte y esta relación tiene atributos, dichos atributos serán eliminados del sistema.

Función	Añadir un atributo a una relación.
Descripción	Añade un nuevo atributo a una relación existente en el proyecto en curso.
Entrada	Vector de 2 o 3 elementos: El primero de ellos es un TransferRelacion que contiene toda la información de la relación a la que se desea añadir el nuevo atributo. El segundo es un TransferAtributo que contiene toda la información introducida por el usuario para ese atributo (nombre, carácter compuesto, carácter multivalorado y dominio). Si el vector tiene un tercer elemento éste es el tamaño del dominio del atributo para aquellos dominios en los que es necesario especificar su tamaño.
Salida	Vector de 2 elementos: El primero de ellos es el TransferRelacion de entrada, con el identificador del nuevo atributo añadido a su lista de atributos.

	El segundo de ellos es un TransferAtributo igual que el de entrada al que el sistema le ha asignado un identificador único.
Origen	GUI_AnadirAtributoRelacion.
Destino	Sistema.
Necesita	DAORelaciones, DAOAtributos.
Acción	Añade a la relación seleccionada un nuevo atributo con las características que desee para éste el usuario.
Precondición	La relación a la que se le quiere añadir un atributo no puede ser de tipo IsA. El nombre que se proporcione para el nuevo atributo no puede ser vacío. Si el dominio del nuevo atributo exige un tamaño, éste debe tener el formato adecuado, esto es, debe ser un valor entero positivo.
Postcondición	En caso de éxito, informar sobre la inserción del nuevo atributo y sobre la modificación de la relación a la que se le ha insertado. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Añadir una entidad a una relación.
Descripción	Añade una entidad como participante en una relación.
Entrada	Vector de 4 elementos: El primero es un TransferRelacion que contiene toda la información de la relación. El segundo es un TransferEntidad con toda la información de la entidad que se quiere añadir a la relación. El tercero es una cadena que representa el inicio del rango de la cardinalidad de la entidad en la relación. El cuarto es una cadena que representa el final del rango de la cardinalidad de la entidad en la relación.
Salida	Vector de 4 elementos: es el mismo que el de entrada salvo en el primer elemento (TransferRelacion) al que se ha añadido la entidad con la cardinalidad (inicio de rango, final de rango).

Origen	GUI_AnadirEntidadARelacion.
Destino	Sistema
Necesita	DAORelaciones.
Acción	Añade una entidad, con una cardinalidad de (inicio, final) a la relación seleccionada.
Precondición	La relación no puede ser de tipo IsA. La cardinalidad de la entidad en la relación debe tener el formato correcto (ambos enteros positivos y el inicio menor o igual que el final de rango).
Postcondición	En caso de éxito, informar sobre la adición de la entidad a la relación y con la cardinalidad que lo hace. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Quitar una entidad de una relación.
Descripción	Quita una entidad participante en una relación.
Entrada	Vector de 2 elementos: El primero es un TransferRelacion con toda la información de la relación. El segundo es un TransferEntidad con toda la información de la entidad que se quiere quitar como participante de relación.
Salida	Vector de 2 elementos: El primero de ellos es un TransferRelacion con la relación IsA de entrada en la que se ha quitado la entidad de entrada como entidad participante. El segundo es un TransferEntidad. Es el mismo que el de entrada.
Origen	GUI_QuitarEntidadARelacion.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Quita la entidad como participante de la relación seleccionada.

Precondición	La relación no debe ser de tipo IsA. La relación debe tener establecida como participante la entidad que se desea quitar.
Postcondición	En caso de éxito, informar sobre el la modificación de la relación, quitando la entidad deseada de las entidades participantes. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Editar la cardinalidad de una entidad en una relación.
Descripción	Modifica la cardinalidad de la entidad en la relación.
Entrada	Vector de 4 elementos: El primero es un TransferRelacion que contiene toda la información de la relación. El segundo es un TransferEntidad con toda la información de la entidad que se quiere añadir a la relación. El tercero es una cadena que representa el nuevo valor para el inicio del rango de la cardinalidad de la entidad en la relación. El cuarto es una cadena que representa el nuevo valor para el final del rango de la cardinalidad de la entidad en la relación.
Salida	Vector de 4 elementos: es el mismo que el de entrada salvo en el primer elemento (TransferRelacion) al que se ha editado la cardinalidad (nuevo inicio de rango, nuevo final de rango) de la entidad, situada en segundo elemento del vector.
Origen	GUI_EditarCardinalidadEntidad
Destino	Sistema
Necesita	DAORelaciones.
Acción	Edita la cardinalidad de la entidad deseada (estableciendo los nuevos valores) a la relación seleccionada.
Precondición	La relación no puede ser de tipo IsA. La cardinalidad de la entidad en la relación debe tener el formato correcto (ambos enteros positivos y el inicio menor o igual que el final de rango).
Postcondición	En caso de éxito, informar sobre la modificación de la cardinalidad de la entidad en la relación y con la cardinalidad que lo hace.

	Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Ninguno.

Función	Eliminar una relación.
Descripción	Eliminar una relación existente en el sistema.
Entrada	TransferRelacion que contiene la relación que se desea eliminar.
Salida	TransferRelacion con la relación que ha sido eliminada.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Elimina una relación del sistema y quita las referencias a las entidades que intervienen en ella.
Precondición	Ninguna
Postcondición	En caso de éxito, información sobre la eliminación de la relación. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Si la relación a eliminar tiene atributos, éstos serán también eliminados del sistema.

2.3.2. Relaciones de herencia IsA.

Los requisitos funcionales correspondientes a este tipo de relaciones de herencia son lo siguientes:

1. Insertar una relación IsA.
2. Establecer la entidad padre en una relación IsA.
3. Quitar la entidad padre de una relación IsA.
4. Añadir una entidad hija a una relación IsA.
5. Quitar una entidad hija de una relación IsA.
6. Eliminar una relación IsA.
7. Mover la posición de una relación.

Función	Insertar una relación IsA.
Descripción	Añadir una nueva relación IsA al proyecto en curso.
Entrada	TransferRelacion la posición en el panel donde se ha pinchado.
Salida	TransferRelacion que contiene la nueva relación IsA que se ha añadido al sistema.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Crea una nueva relación de herencia IsA en el proyecto en curso.
Precondición	Ninguna
Postcondición	En caso de éxito, información sobre la inserción de la nueva relación IsA. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Establecer la entidad padre de una relación IsA
Descripción	Establece la entidad padre en una relación de herencia.
Entrada	Vector de 2 elementos: El primero de ellos es un TransferRelacion con toda la información de la relación IsA. El segundo es un TransferEntidad con toda la información de la entidad que se quiere establecer como entidad padre.
Salida	Vector de 2 elementos: El primero de ellos es un TransferRelacion con la relación IsA de entrada en la que se ha definido la entidad de entrada como entidad padre. El segundo es un TransferEntidad. Es el mismo que el de entrada.
Origen	GUI_EstablecerEntidadPadre.
Destino	Sistema.

Necesita	DAORelaciones
Acción	Establece para la relación seleccionada la entidad deseada como entidad padre de la relación.
Precondición	La relación debe ser de tipo IsA. La relación no debe tener establecida previamente ninguna entidad padre. Debe existir alguna entidad en el sistema.
Postcondición	En caso de éxito, informar sobre el establecimiento de la entidad como entidad padre de la relación IsA. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Quitar la entidad padre de una relación IsA
Descripción	Quita la entidad padre de una relación de herencia.
Entrada	TransferRelacion con toda la información de la relación IsA.
Salida	TransferRelacion con la relación IsA de entrada en la que se ha quitado la entidad que tenía como entidad padre.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Quita para la relación seleccionada la entidad que tenía definida como entidad padre.
Precondición	La relación debe ser de tipo IsA. La relación debe tener establecida previamente una entidad padre.
Postcondición	En caso de éxito, informar sobre el la modificación de la relación IsA. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Al quitar la entidad padre de una relación IsA también serán quitadas las referencias a las posibles entidades hijas que tenga.

Función	Añadir una entidad hija a una relación IsA
Descripción	Añade una entidad como entidad hija en una relación de herencia.
Entrada	Vector de 2 elementos: El primero de ellos es un TransferRelacion con toda la información de la relación IsA. El segundo es un TransferEntidad con toda la información de la entidad que se quiere establecer como entidad hija.
Salida	Vector de 2 elementos: El primero de ellos es un TransferRelacion con la relación IsA de entrada en la que se ha añadido la entidad de entrada como entidad hija. El segundo es un TransferEntidad. Es el mismo que el de entrada.
Origen	GUI_AnadirEntidadHija.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Añade una entidad como entidad hija en una relación de herencia IsA.
Precondición	La relación debe tener establecida previamente la entidad padre. Debe existir alguna otra entidad en el sistema, sin contar la entidad padre.
Postcondición	En caso de éxito, informar sobre el establecimiento de la entidad como entidad hija de la relación IsA. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Quitar una entidad hija de una relación IsA
Descripción	Quita una entidad hija de una relación de herencia.
Entrada	Vector de 2 elementos: El primero es un TransferRelacion con toda la información de la relación IsA. El segundo es un TransferEntidad con toda la información de la entidad que se quiere quitar como entidad hija de la relación.
Salida	Vector de 2 elementos:

	El primero de ellos es un TransferRelacion con la relación IsA de entrada en la que se ha quitado la entidad de entrada como entidad hija. El segundo es un TransferEntidad. Es el mismo que el de entrada.
Origen	GUI_QuitarEntidadHija.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Quita la entidad que se desee como hija de la relación de herencia seleccionada.
Precondición	La relación debe ser de tipo IsA. La relación debe tener establecida previamente una entidad padre. La entidad que se desea quitar debe ser una entidad hija de la relación IsA.
Postcondición	En caso de éxito, informar sobre el la modificación de la relación IsA al quitando la entidad deseada de las entidades hijas. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

Función	Eliminar una relación IsA.
Descripción	Eliminar una relación IsA existente en el sistema.
Entrada	TransferRelacion que contiene la relación IsA que se desea eliminar.
Salida	TransferRelacion con la relación IsA que ha sido eliminada.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Elimina una relación IsA del sistema y quita las referencias a las entidades que intervienen en ella.
Precondición	Ninguna
Postcondición	En caso de éxito, información sobre la eliminación de la relación IsA. Si no se ha podido realizar, información explicativa sobre la razón.

Efectos laterales	Ninguno.

Función	Mover la posición de una relación.
Descripción	Mueve la relación seleccionada de una posición a otra en el diagrama E/R.
Entrada	TransferRelacion con toda la información de la relación a la que se desea cambiar su posición en el diagrama. La nueva posición ya está setteada en el TransferRelacion.
Salida	TransferRelacion a la que se le ha modificado su atributo posición.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Cambia el atributo posición de la relación seleccionada a la nueva posición.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre el cambio de posición de la entidad. Si no se ha podido realizar, información explicativa sobre la razón de ello.
Efectos laterales	Ninguno.

2.4. Módulo Sistema.

Los requisitos funcionales correspondientes al sistema son los siguientes:

1. Validación del diseño.
2. Generación del Modelo Relacional
3. Generación del Script SQL.
4. Generación del fichero .sql.

Función	Validación del diseño
Descripción	Valida el diseño E/R de acuerdo a las reglas de diseño de BD.
Entrada	Ninguna.

Salida	Secuencia de cadenas informativas sobre el proceso de validación.
Origen	GUI_Principal
Destino	GUI_Principal
Necesita	DAOEntidades, DAOAtributos y DAORelaciones.
Acción	Realiza la validación del diseño realizado e informa de los resultados al usuario. Nota: Para más información sobre los criterios de validación consultar la sección de teoría.
Precondición	Ninguna.
Postcondición	Se muestra una secuencia de mensajes correspondientes a los resultados de la validación.
Efectos laterales	Ninguna.

Función	Generación del Modelo Relacional
Descripción	Genera el Modelo Relacional derivado del diseño realizado siguiendo las restricciones especificadas en éste.
Entrada	Ninguna.
Salida	Modelo Relacional en formato texto.
Origen	GUI_Principal
Destino	GUI_Principal
Necesita	DAOEntidades, DAOAtributos y DAORelaciones.
Acción	Genera el modelo relacional derivado del diseño. Nota: Para más información sobre los criterios de generación consultar la sección de teoría.
Precondición	Diseño previamente validado de forma satisfactoria.
Postcondición	Se muestra al usuario el modelo relacional generado.
Efectos laterales	Ninguno.

Función	Generación del Script SQL
Descripción	Genera el código SQL asociado al diseño realizado.
Entrada	Ninguna.
Salida	Código SQL en formato texto.
Origen	GUI_Principal
Destino	GUI_Principal
Necesita	DAOEntidades, DAOAtributos y DAORelaciones.
Acción	Genera el código SQL asociado al diseño realizado
Precondición	Diseño previamente validado de forma satisfactoria.
Postcondición	Se muestra al usuario el código SQL generado.
Efectos laterales	Ninguno.

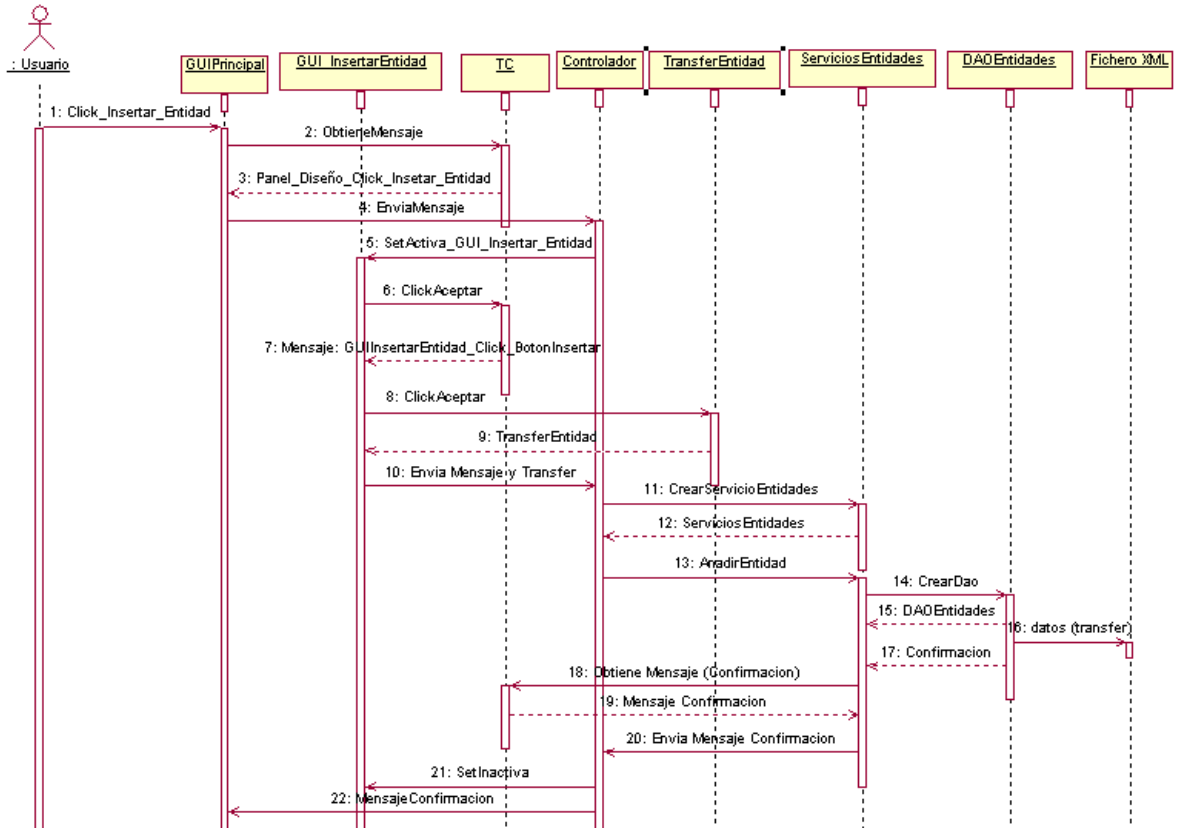
Función	Generación del fichero SQL
Descripción	Vuelca el código SQL generado a un fichero .sql.
Entrada	Ninguna.
Salida	Fichero .sql con el script del diseño.
Origen	GUI_Principal
Destino	GUI_Principal
Necesita	Nada.
Acción	Guarda en un fichero externo .sql el código SQL generado.
Precondición	Diseño previamente validado de forma satisfactoria y script SQL ya generado.
Postcondición	Fichero .sql correctamente creado.
Efectos laterales	Ninguno.

3. Diagramas de secuencia.

Hemos querido incluir los diagramas de secuencia de cuatro de los casos de uso mas frecuentes en la aplicación. De este modo esperamos que el lector pueda comprender más fácilmente el funcionamiento interno de ésta.

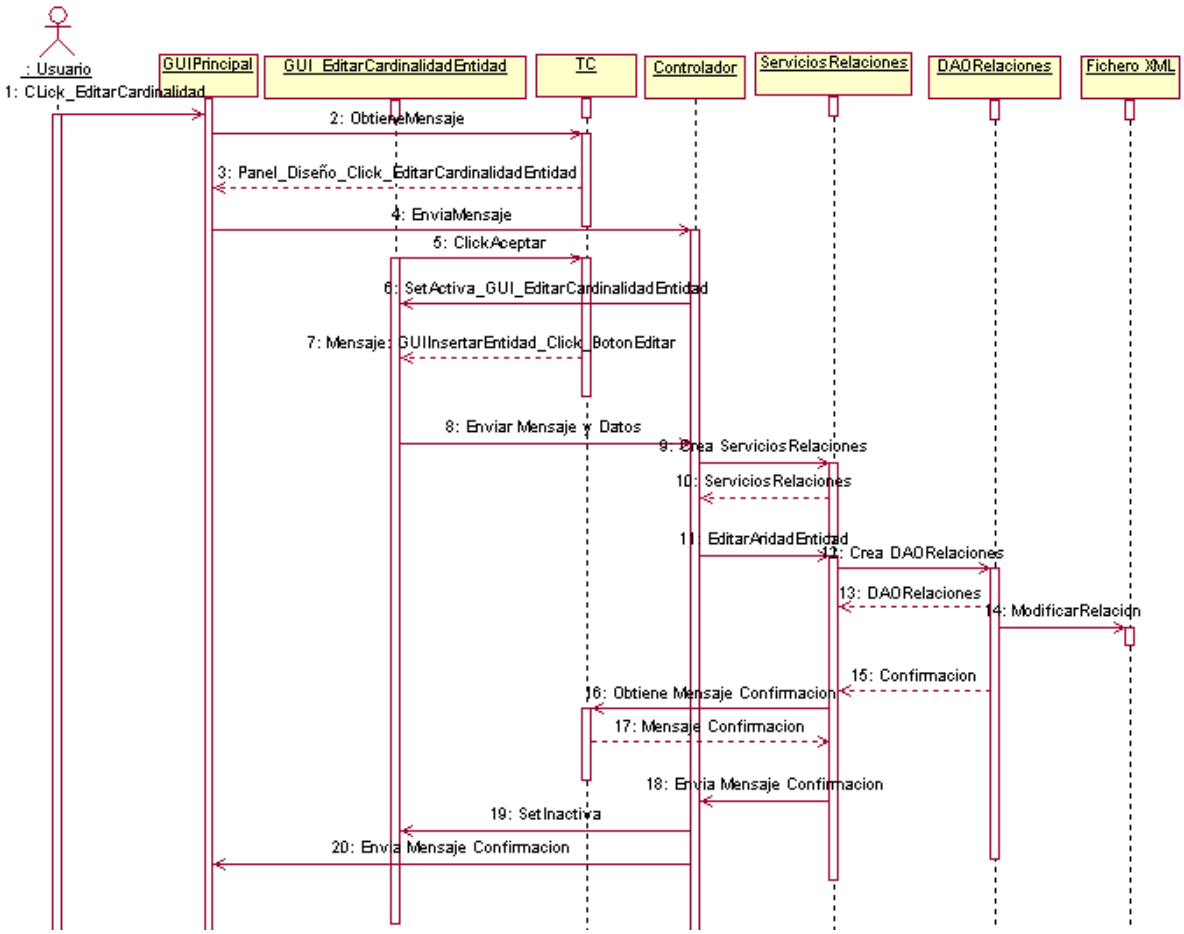
3.1. Insertar una entidad.

El proceso comienza al pulsar el comando para *Insertar una entidad*. A continuación la interfaz principal accede al *Tipo Controlar* solicitando el mensaje correspondiente a una inserción de una entidad. Se envía dicho mensaje al *Controlador* que activa la vista particular para la operación. Tras introducir los datos, se crea un *Transfer* que los contenga. Así mismo se llama a los *Servicios de Entidades* y proporcionándole el *Transfer* se ejecuta el método asociado a la inserción de entidades. Se crea un *DAO* para acceder al fichero *XML* y se inserta la nueva entidad devolviendo una confirmación que generará nuevos mensajes también de confirmación para el controlador.



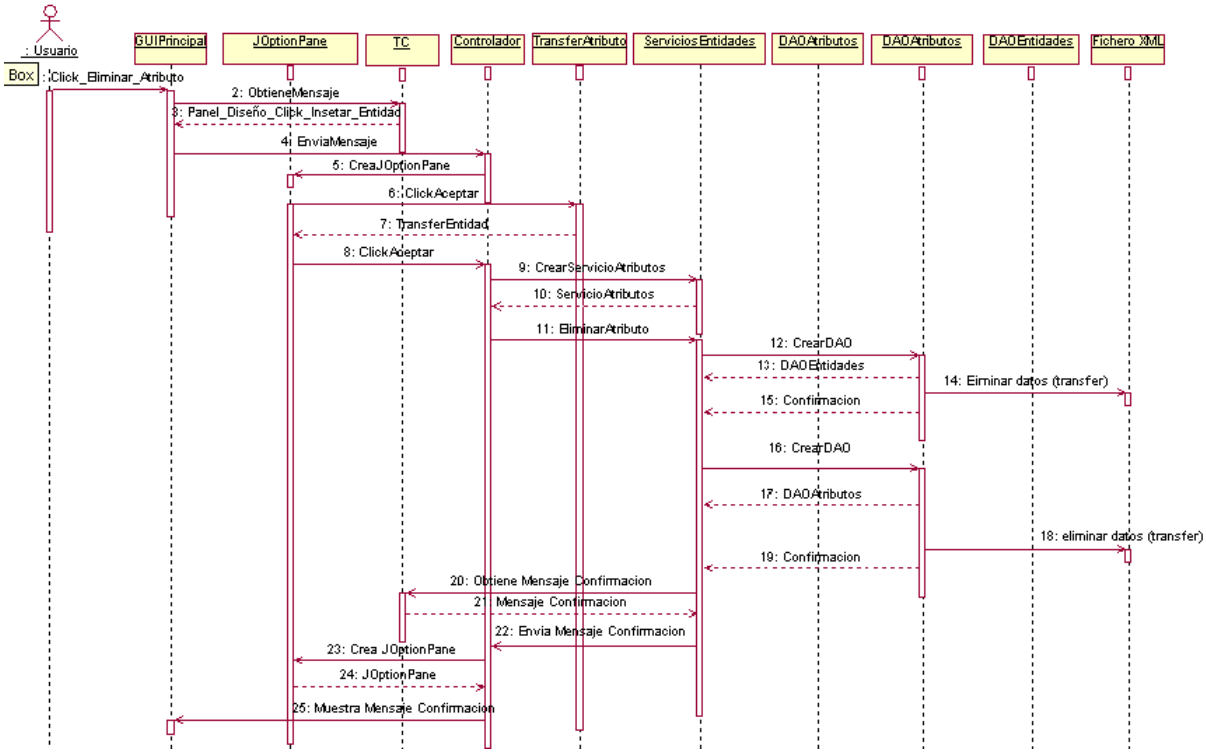
3.2. Editar la cardinalidad de una entidad en una relación.

El proceso comienza al pulsar el comando para *Editar la aridad de una entidad en una relación*. A continuación la interfaz principal accede al *Tipo Controlar* solicitando el mensaje correspondiente a una edición de cardinalidades. Se envía dicho mensaje al *Controlador* que activa la vista particular para la operación. Se envían los datos a los *Servicios de Relaciones* previamente creados y se modifican la información mediante el *DAO* de *relaciones.xml*, devolviendo un mensaje de confirmación que se propaga por las capas.



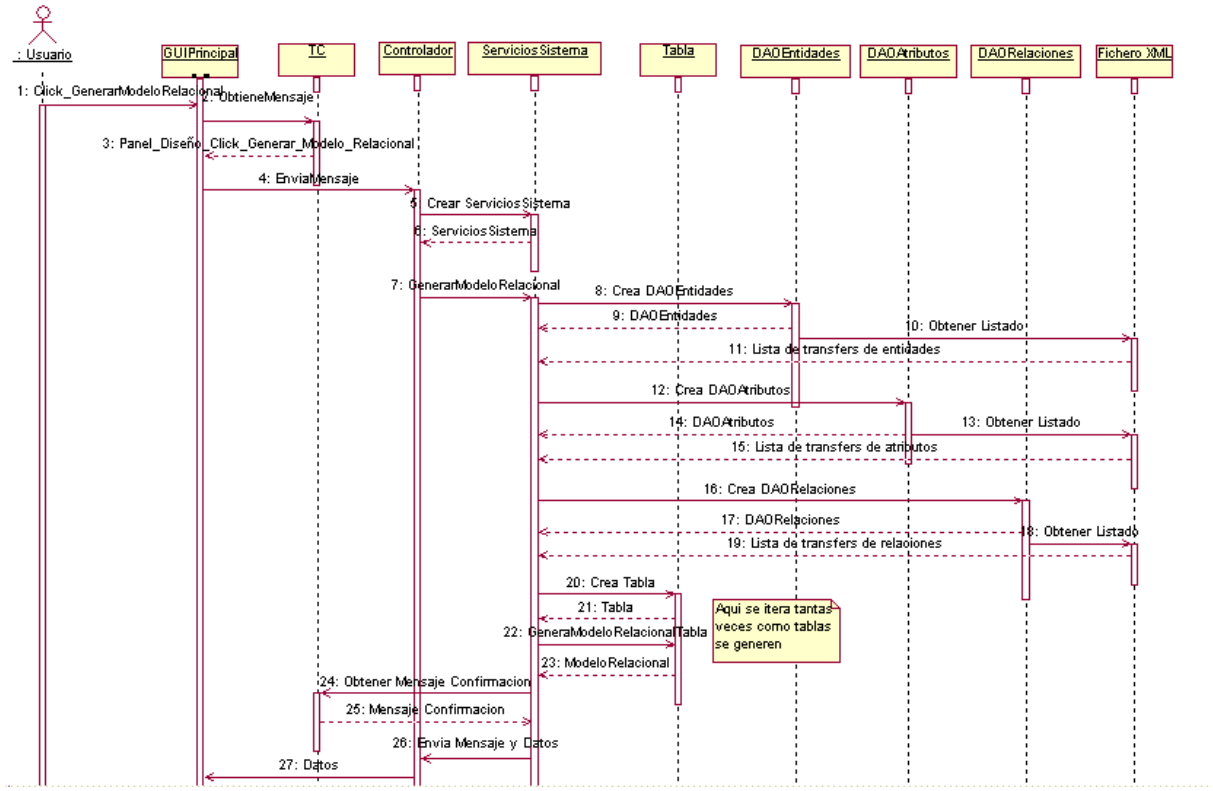
3.3. Eliminar Atributo de una entidad.

El proceso comienza al pulsar el comando para *Eliminar un atributo*. A continuación la interfaz principal accede al *Tipo Controlar* solicitando el mensaje correspondiente a una eliminación de un atributo. Se envía dicho mensaje al *Controlador* que activa un mensaje informativo que pide la confirmación del usuario para llevar a cabo la operación. Después se usa el *Servicio de Atributos* y los *DAOs de Entidades y Atributos*, ya que se deben modificar ambos ficheros. Una vez realizada la eliminación, se envía un mensaje de confirmación que se propaga por las capas de la aplicación.



3.4. Generar el Modelo Relacional.

El proceso comienza al pulsar el comando para *Generar el Modelo Relacional*. A continuación la interfaz principal accede al *Tipo Controlar* solicitando el mensaje correspondiente a dicha generación. Se envía dicho mensaje al *Controlador* y éste usa los *Servicios de Sistema*. Este servicio crea *DAOs* para acceder a los tres ficheros *XML* y obtener los listados de objetos. Crea también objetos de la clase *Tabla* iterativa mente por cada conjunto que lo requiere. Para cada objeto *Tabla* solicita el Modelo Relacional correspondiente y envía el conjunto total como respuesta a los *Servicios de Sistema*. A continuación, mediante un mensaje de confirmación del *Tipo Controlar*, se llega a la capa de presentación donde se muestra el modelo relacional obtenido.



Arquitectura del sistema.

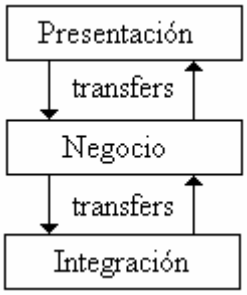
1. Arquitectura multicapa.

DBDT tiene una estructura multicapa que permite tener dividida la aplicación en diferentes unidades funcionales totalmente independientes. Esto asegura una división clara de responsabilidades y hace que el sistema sea más mantenible y extensible.

Se distinguen tres capas fundamentales: presentación, negocio e integración:

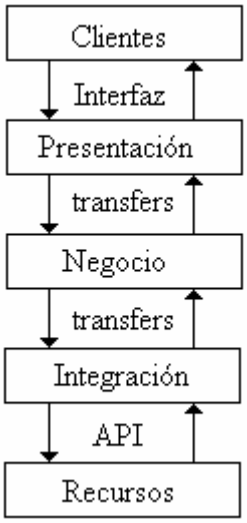
- La capa de presentación encapsula toda la lógica de presentación necesaria para dar servicios a los usuarios que utilizan el sistema.
- La capa de negocio proporciona los servicios del sistema.
- La capa de integración es la responsable de la comunicación con los recursos y sistemas externos.

Una representación gráfica de una arquitectura multicapa de es la siguiente:



En realidad se trata de una arquitectura de cinco capas, ya que incluye las capas de clientes y de recursos:

- La capa de clientes representa a todos los usuarios que acceden al sistema. Se encuentra situada por encima de la capa de presentación.
- La capa de recursos contiene todos los datos del negocio y los recursos externos. Está por debajo de la capa de integración.



El uso de una arquitectura multicapa ofrece múltiples ventajas frente a una arquitectura de una sola capa o de dos capas. Las principales ventajas son la modularidad y la adaptabilidad.

Cada una de las capas se comporta como una caja negra: ofrece una interfaz a la capa inferior y se comunica con la capa superior a través de la que ésta le proporciona. Debido a esto, se pueden modificar cada una de las capas por separado sin afectar a las demás.

Por el contrario, tiene el inconveniente de tener una mayor complejidad arquitectónica. Sin embargo, esta desventaja se ve superada con creces por las ventajas que proporciona.

Para la implementación de la arquitectura multicapa descrita anteriormente, nos han sido especialmente útiles los siguientes patrones arquitectónicos:

1. Modelo Vista Controlador (MVC).
2. Patrón Transferencia (Transfer).
3. Patrón Objeto de Acceso a Datos (DAO)

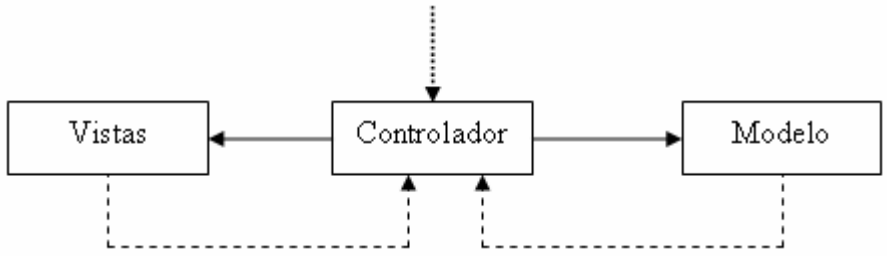
A continuación describiremos cada uno de estos patrones y explicaremos la motivación que nos ha llevado a usarlos.

2. Patrones de diseño.

2.1. Modelo Vista Controlador (MVC)

El modelo Vista Controlador (MVC) es un patrón de arquitectura software que divide una aplicación interactiva en tres capas fundamentales:

1. El **modelo** contiene la funcionalidad básica. En nuestro caso, contiene los servicios de aplicación y la persistencia de los datos:
    1. Los servicios de aplicación implementan todos los requisitos software del sistema.
    2. La persistencia se encarga de la gestión explicita de los datos de nuestra aplicación.
2. Las **vistas** tienen una doble funcionalidad: muestran y recogen información al/del usuario.
3. El **controlador** media entre las vistas y el modelo.



De las dos variantes que tiene el MVC, DBDT usa la versión con controlador **activo**: cualquier cambio en el modelo es notificado por el controlador a las vistas.

Toda la comunicación multicapa del sistema pasa por el controlador, es decir, la única forma de comunicar las vistas y el modelo es a través de éste.

De forma esquemática, el flujo de control que sigue este modelo es el siguiente:

2. El usuario interactúa con la interfaz de usuario de alguna forma, por ejemplo, pulsando uno de los botones.
3. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega.
4. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario.
5. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo (pasando previamente por el controlador) para generar la interfaz apropiada para el usuario, donde se reflejan los cambios realizados en el modelo. El modelo no tiene conocimiento directo sobre la vista.
6. En nuestra implementación, la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.
7. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Este modelo ofrece múltiples ventajas. Entre ellas podemos destacar las siguientes:

- posibilita tener un modelo independiente de la representación de la salida y del comportamiento de la entrada,
- permite tener simultáneamente múltiples vistas para un mismo modelo.

Su interés de uso principal radica en la independencia de los cambios: se pueden hacer las modificaciones que se crean oportunas en cada una de las capas de la aplicación y no afectar al resto del modelo.

2.2. Patrón Transferencia (Transfer).

El patrón transferencia, más conocido como transfer, es un patrón de arquitectura software cuyo propósito es independizar el intercambio de datos entre capas.

Dado que nuestro objetivo es independizar cada una de las capas de nuestro sistema de las demás, no es posible que una capa tenga conocimiento de la representación de las entidades dentro de las otras.

Su utilidad radica en el hecho de homogeneizar el intercambio de datos entre las capas, de modo que en cada una de las transferencias de información solamente se intercambian objetos serializables: objetos transfer, conjuntos de transfer y otros objetos serializables (enteros, cadenas, caracteres...).

DBDT usa tres tipos de transfer, uno para cada una de las entidades con las que trabaja nuestro sistema. Son los siguientes:

- TransferEntidad.
- TransferAtributo.
- TransferRelacion.

El acceso a cada uno de los objetos transfer se realiza por medio de getters y de setters. De modo ilustrativo, mostramos el contenido de uno de los transfers:

```
public class TransferEntidad extends Transfer{
    private int idEntidad;
    private String nombre;
    private boolean debil;
    private Vector listaAtributos;
    private Vector listaClavesPrimarias;
    private Point2D posicion;
```

Si se desea conocer la implementación concreta de cada uno de los transfers, consultar la sección de implementación de esta memoria.

La ventaja de trabajar con objetos transfers es su propia definición: independizar el intercambio de datos entre capas. Con ellos promovemos la modularidad del sistema y la independencia entre capas del mismo.

2.3. Objeto de Acceso a Datos (DAO)

El patrón DAO, se puede incluir en un diseño de una aplicación multicapa en la capa de persistencia.

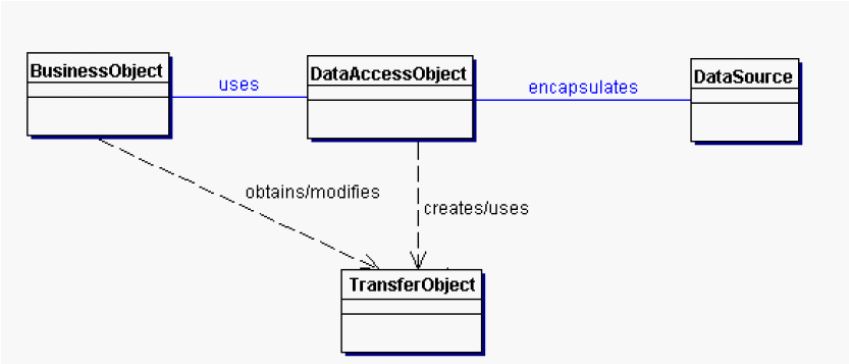
El objetivo básico del patrón es facilitar y homogeneizar el acceso a los datos de la aplicación. Estos datos pueden tener una estructura concreta que se refleja en un sistema determinado de representación. En nuestro sistema, se trata de una colección de ficheros XML.

La idea de controlar los datos nos obliga a adquirir ciertos conocimientos sobre el sistema que los contendrá. Para conocer más sobre el acceso a los ficheros XML puede consultarse la sección de la documentación encargada de la implementación de la capa de persistencia.

La independencia del resto de capas que supone el uso del patrón DAO permite modificar precisamente el modo de almacenaje de los datos, pudiendo variar de un fichero a una base de datos o cualquier otra estructura. Esto permite optimizaciones en el diseño de cualquier aplicación. El único inconveniente que se le puede encontrar al patrón DAO puede ser que aumenta el número de objetos del sistema.

Las operaciones esenciales de un DAO son la escritura, la modificación, la consulta y la eliminación de datos. DBDT, además de implementar estas cuatro operaciones en cada uno de los DAOs, incluye la posibilidad de obtener un listado con todos los objetos de información.

Cabe destacar que los objetos que se manejan en cada uno de nuestros DAOs son objetos Transfer o conjuntos de ellos.



Implementación del sistema.

1. Introducción.

Para la implementación final de la aplicación se ha escogido el lenguaje de programación **Java**.

Bien podría haberse escogido otro lenguaje de mayor rendimiento como C++, a priori, pero, al ser uno de los objetivos prioritarios la aplicación docente de la herramienta, se ha considerado como mejor alternativa el lenguaje definido originalmente por Sun Microsystems.

Es cierto que en torno al 90% de los sistemas de escritorio poseen un sistema operativo Windows, de la empresa Microsoft, pero en entornos académicos, y especialmente aquellos a los que se ha considerado más práctica la herramienta, dichos porcentajes varían considerablemente. En este motivo principal se basa la elección del lenguaje Java, y su correcto funcionamiento ha sido comprobado en sistemas Windows y distribuciones GNU/Linux (Debian y Ubuntu).

El diseño de la aplicación trata de ser, en todo lo posible, independiente de la arquitectura y el sistema operativo utilizado, para que cualquier usuario tenga la capacidad de utilizarlo para el fin que estime oportuno.

La máquina virtual de Java utilizada en todos los casos ha sido la ofrecida por Sun Microsystems versión 1.6.0 (o posteriores). La elección de dicha máquina está motivada por la mejora técnica de la misma frente a versiones previas y funciones que la misma posee de forma única.

2. Organización de paquetes.

La implementación de la herramienta se estructura en una serie de paquetes, en los que se modularizan todos los comportamientos y acciones de la misma, para una fácil reusabilidad de código y simplicidad de comprensión del mismo.

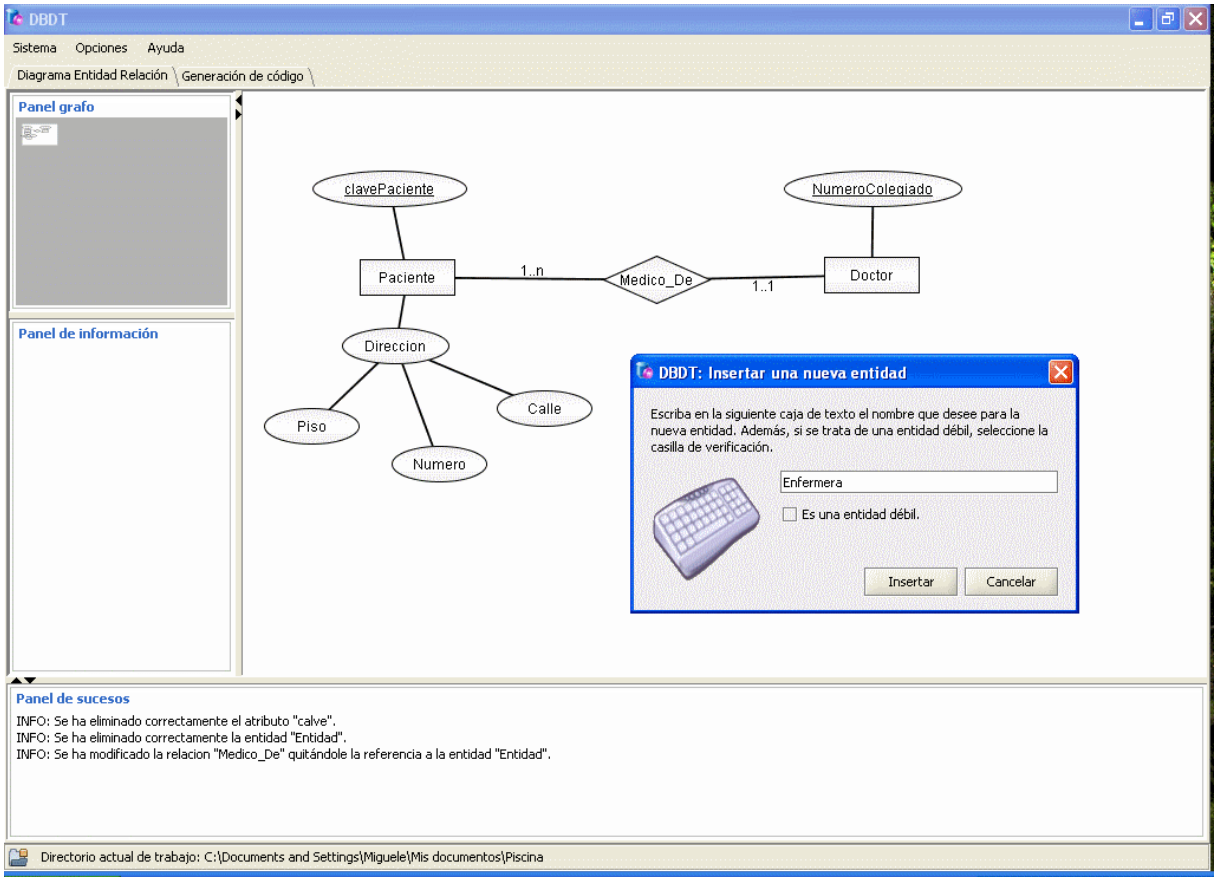
A continuación se presenta una breve descripción de los mismos:

- **Controlador**. En este paquete se describen tanto el comportamiento del módulo Controlador como los mensajes que el mismo es capaz de enviar y recibir.
- **LogicaNegocio**. Este paquete contiene dos subpaquetes internos en los que especifica, de forma más concreta la lógica de la aplicación:
  - **Servicios**. Proporciona a la aplicación el interfaz y ejecución de los servicios lógicos ofrecidos por la misma. Dichos servicios serán abstractos para el resto de módulos de la herramienta y realizarán, de forma atómica, las acciones necesarias para su correcto funcionamiento.
  - **Transfers**. Define las clases que modelarán los objetos en el flujo de mensajes de la aplicación. Los objetos Transfer son aquellos que se transmiten de módulo en módulo, a partir de mensajes, y en los que se almacena cualquier cambio o acción que ha de ser tratada.
- **Persistencia**. En este paquete se definen las clases que trabajan a más bajo nivel. DataBase Design Tool es una aplicación que trabaja con datos persistentes en todo momento y las clases DAOS, que este paquete contiene, son las encargadas de su correcto funcionamiento al nivel de fichero en disco.
- **Presentación**. Este paquete contiene el conjunto de GUI (Interfaz Gráfica de Usuario) que el usuario de la aplicación visualizará en su ejecución. Todas ellas, a excepción de la principal, poseen una estructura similar o cuasi-idéntica, para facilitar la usabilidad de la aplicación. Es importante mencionar que las interfaces únicamente tienen la tarea de mostrar datos y recibir acciones por parte del usuario, y nunca realizar gestiones con los mismos.
  El paquete Presentación posee un subpaquete interno, para mejor modularidad, que contiene de forma separada la representación gráfica del esquema:
  - **Grafo**. En este paquete se describen las clases necesarias para la visualización del grafo relacional presentado por la aplicación. Para dicha visualización se ha hecho uso de la librería JUNG (Java Universal Network/Graph Framework), habiendo sido necesario una redefinición de muchas de las características que dicha librería ofrece.
- **Utilidades**. Este último paquete constituye aquellas partes de la herramienta que han sido utilizadas como elementos externos o no-propios del diseño. Dichas partes, entre otras, son: un lanzador de ayuda mediante el explorador Web propio del sistema nativo, rutas de las imágenes presentadas en la aplicación, utilidades sobre código HTML...

3. Presentación.

La Presentación de la aplicación engloba todas las interfaces de usuario existentes, además del diseño que representará el esquema relacional.

El conjunto de interfaces está constituido por un *frame*, o marco principal, un diálogo de cambio de espacio de trabajo y dieciséis interfaces de acción/decisión. El marco de diseño gráfico, contenido en el frame, será expuesto de forma singular, al ser su implementación la más crítica dentro de esta capa.



El objetivo principal, y único, de todas las interfaces de usuarios es ser el nexo entre el usuario y los servicios ofrecidos por el sistema. De esta forma, la interfaz no manipula ni realiza cambio alguno en ningún dato, sino que registra las variaciones emitidas por el usuario de la aplicación y, si es necesario, actualiza la vista de los datos con la información obtenida de los demás módulos.

Por lo tanto, el interfaz de todas las GUIs (Graphic User Interface) consiste básicamente en recopilar eventos de acción del usuario, ya sea mediante pulsaciones de botones, activaciones de campos en menús desplegables o el mero hecho de usar la rueda del ratón, por ejemplo.

Todas las interfaces son generadas de forma estática al comienzo de la aplicación. Por este motivo, todas ellas, poseen los métodos:

```
public void setActiva()
public void setInactiva()
```

El nombre de los métodos es, explícitamente, lo que realiza su implementación. Si se activa un interfaz se muestra, mientras que si se desactiva sencillamente se oculta, no se libera su memoria. Este funcionamiento se expondrá de forma más concisa en el desarrollo de la implementación del Controlador.

Para simplificar el desarrollo únicamente se expondrá el esqueleto de una interfaz de acción/decisión, dado que, a excepción de ciertos contenidos gráficos como cajas de texto o combos de opciones, todas comparten el mismo esquema.

3.1 Interfaces de acción/decisión.

Estas interfaces se mostrarán cuando el usuario realice una acción en la que se necesite completar ciertos parámetros adicionales o, ya bien, solicitar una confirmación por parte del mismo.

Acciones pueden ser, entre otras: insertar una entidad en el diagrama, insertar una relación, añadir un atributo a una relación, alterar los parámetros de un atributo...

Por lo tanto, el esquema de todo interfaz acción/decisión será el de ofrecer ciertas propiedades susceptibles de cambio y solicitar al usuario la confirmación por medio de dos botones: **Aceptar** y **Cancelar**.

Si el usuario selecciona el botón de **Cancelar**, la interfaz se ocultará y no se realizará modificación alguna en los datos. No existe ninguna otra acción asociada a dicho botón, pues no es necesaria.

Por el contrario, si el usuario selecciona el botón de **Aceptar**, la interfaz recogerá todos los datos contenidos en la misma, que el usuario habrá completado o rellenado, y enviará un mensaje al Controlador notificando el tipo de acción que se ha llevado a cabo y almacenando en un objeto serializable (Transfer, conjunto de Transfers, cadena, entero…) la información asociada al cambio en los datos.

Se adjunta un ejemplo, en el caso de inserción de una entidad en el esquema, como ejemplo de implementación. Al poseer el mismo esquema todas las interfaces, no será necesario exponer casos excepcionales.

```
private void botonInsertarActionPerformed(java.awt.event.ActionEvent evt) {
    // Generamos el transfer que mandaremos al controlador
    TransferEntidad te = new TransferEntidad();
    te.setPosicion(this.getPosicionEntidad());
    te.setNombre(this.cajaNombre.getText());
    te.setDebil(this.CasillaEsDebil.isSelected());
    te.setListaAtributos(new Vector());
    te.setListaClavesPrimarias(new Vector());
    // Mandamos mensaje + datos al controlador
    this.getControlador().mensajeDesde_GUI(TC.GUIInsertarEntidad_Click_BotonInsertar, te);
}
```

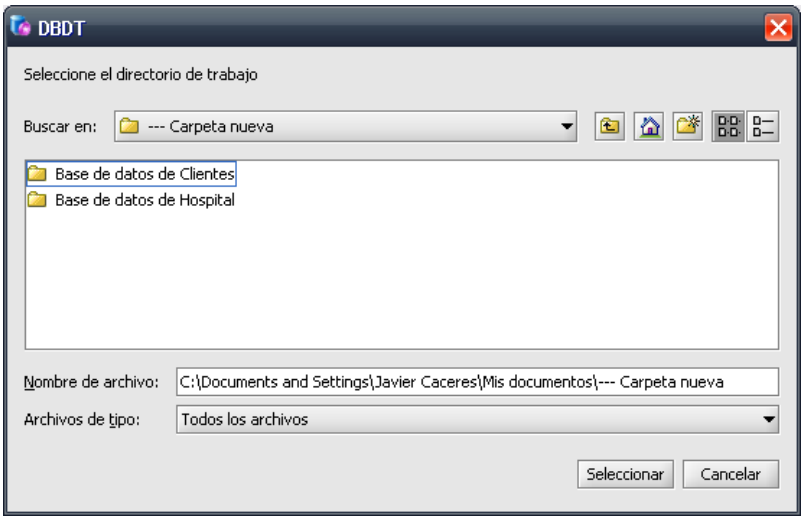
Al separar responsabilidades por módulos, o capas, no es necesario un mayor desarrollo en las acciones de usuario. Tras el envío de este mensaje, será el Controlador el encargado de ocultar la interfaz y completar los datos en el frame principal.

3.2. Diálogo de cambio de espacio de trabajo (Workspace).

Esta interfaz se expone de forma separada al no compartir la funcionalidad de todas las demás.

La tarea de la misma es simple: permitir al usuario cambiar el directorio de trabajo a otro para proseguir con su desarrollo.

El espacio de trabajo será el directorio en el que se almacenan los datos de forma persistente y, por lo tanto, el usuario ha de poder alterar el espacio de trabajo para poder desarrollar en paralelo distintos proyectos.



Debido a las posibles situaciones resultantes de la elección de directorio, por medio de un cuadro de selección de directorio propio del sistema nativo, esta interfaz ha de ser capaz de realizar dos tareas diferenciadas:

Crear un nuevo espacio de trabajo. Si el usuario escoge un directorio en el que no existen datos con los que partir para el desarrollo de un proyecto, esta interfaz preguntará al usuario si desea generar un nuevo proyecto en el directorio seleccionado.
Se recomienda que un espacio de trabajo esté situado en un directorio independiente, sin mezcla con otro tipo de ficheros, para la seguridad de la información, pero si el usuario desea utilizar un directorio común se comunicará una confirmación de dicha elección.
Si todos los permisos son concedidos, este interfaz generará los tres ficheros de datos del proyecto con contenido vacío. El contenido vacío hace referencia al contenido lógico, y no al físico, ya que se han de generar cabeceras de ficheros XML y datos iniciales para la aplicación.
Una vez generado el contenido de los ficheros de datos, se notificará al Controlador la generación del espacio de trabajo, y será dicho módulo el que inicialice y active todos los demás objetos con la información del espacio actual.

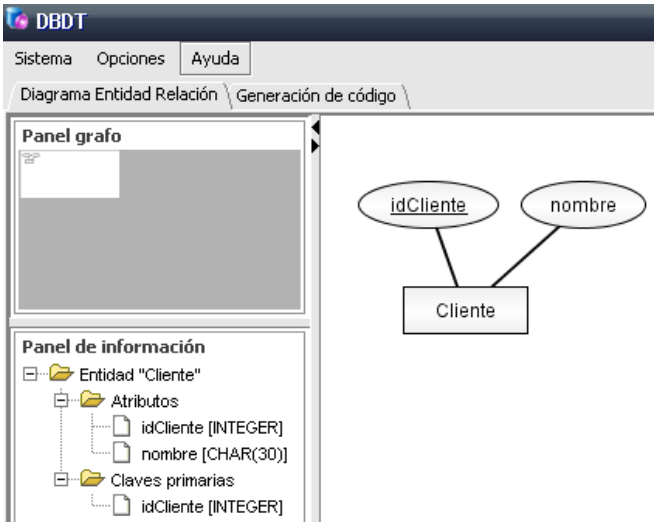
Establecer un espacio de trabajo ya creado. Si el usuario escoge un directorio en el que existen datos de un proyecto de la aplicación, se notificará al módulo Controlador de la ruta de dicho directorio, para que pueda controlar e inicializar todos los módulos de una forma ordenada.

Por lo tanto, el diálogo de cambio de espacio de trabajo posee una funcionalidad bastante reducida, pero suficientemente propia como para ser separable de todas las demás.

3.3. Frame Principal.

Esta interfaz es la GUI principal a la que tendrá acceso el usuario.

Presenta diversas opciones, tanto en menús de título, como en acciones dentro del marco de diseño, y su implementación es tanto más compleja, como más amplia, que el resto de interfaces juntas.



Al seguir el patrón de diseño modelo vista-controlador la descripción de la implementación de esta interfaz se dividirá en dos secciones: las acciones entrantes y las acciones salientes.

Se considerará una acción entrante a aquella que llegue desde componentes internos de la aplicación, desde el Controlador en este caso, y acciones salientes serán aquellas que sean invocadas desde el usuario y realicen peticiones salientes al resto de la herramienta.

NOTA: Los paneles de diseño serán expuestos en una sección separada, dada su complejidad.

A continuación se describen las dos secciones del frame principal:

- **Acciones salientes**. Las acciones salientes son las provocadas por eventos del usuario. Principalmente, y a excepción de las relacionadas con los paneles de diseño, serán llevadas a cabo a través del menú de Opciones o a través de los botones situados en la pestaña Generación de código.
    - **Menú de opciones**. El menú de opciones consta de tres cajones básicos.
        1. **Sistema**. El cajón Sistema posee las opciones de *Cambiar espacio de trabajo* y *Salir*. Ambas selecciones limitan su implementación a notificar al Controlador de la acción deseada por el usuario.
        2. **Opciones**. El cajón Opciones posee las selecciones *de Exportar el diagrama a un fichero gráfico* e *Imprimir el diagrama*.
           Por la estructura, y el diseño, de ambas opciones y dado que no es necesaria la intervención de datos externos se ha elegido implementar la funcionalidad de ambas opciones dentro de la clase del panel de diseño. Ello conlleva que ningún otro módulo es consciente de la acción de exportar o imprimir llevada a cabo, y su funcionamiento se expondrá en el desarrollo del panel de diseño.

3. **Ayuda**. El cajón de Ayuda contiene las opciones de *Contenidos* y *Acerca de DBDT*. Ambas acciones se implementan de manera atómica en el frame principal.
   Contenidos, usando una utilidad externa, abre el Manual de Usuario en formato HTML en el explorador Web por defecto del sistema y Acerca de DBDT se limita a mostrar un diálogo con la información característica de la aplicación.

- **Botones de Generación de Código**. Existen cinco botones de Generación de Código disponibles en la interfaz principal. A excepción del botón Limpiar pantalla de texto, todos los demás se implementan como hilos concurrentes dentro de la aplicación. La motivación de dicha implementación viene derivada del refresco deseado de información en el panel de texto y la no congelación de dicho refresco hasta la finalización de la acción deseada.
  1. *Limpiar pantalla de texto*. Este botón elimina el contenido, en formato texto, que el área de texto de generación posee. No requiere de ningún interfaz externo, pues no manipula ningún tipo de dato.
  2. *Validar modelo relacional*. Dicho botón activa el chequeo del esquema diseñado por el usuario. La única acción que realiza es el envío de un mensaje al módulo Controlador para la comprobación de dicha validación.
  3. *Representación del modelo relacional*. Este botón solicita la representación del modelo relacional por parte de los servicios de la herramienta. Su único interfaz es el envío de un mensaje al módulo Controlador para que se realice dicha acción.
  4. *Generación del script SQL*. Este botón solicita la generación del script SQL en el panel de texto para su posible uso en una base de datos comercial. Su único interfaz es el envío de un mensaje al módulo Controlador para que se realice dicha acción.
  5. *Exportar script a fichero de texto*. Este botón solicita la exportación del script SQL a un fichero de texto ASCII con el objetivo de su futuro uso en una base de datos. Su único interfaz es el envío de un mensaje al módulo Controlador para que se realice dicha acción.

- **Acciones entrantes**. Las acciones entrantes son aquellas en las que se solicita un cambio de contenidos por parte de módulos internos de la aplicación. Como todos los mensajes son centralizados, por parte del módulo Controlador, se ha de seleccionar, en el caso adecuado, la acción que se ha de realizar. Todos los mensajes provenientes de acciones internas modifican, de algún modo, componentes gráficos y, en esencia, suelen modificar los paneles de diseño de la herramienta.
  Por lo tanto, el esquema de implementación patrón para todos los casos es la recepción del mensaje por parte del objeto Controlador, la extracción de los datos necesarios y la actualización en los objetos gráficos de su contenido.

3.4. Marcos de diseño.

Los *marcos de diseño*, como tal, forman parte del frame principal que la aplicación posee, pero se ha separado su exposición debido a la criticidad de los mismos y su notable diferente implementación con respecto a las demás.

Existen tres marcos de diseño en la aplicación DBDT:

1. El Panel de Diseño.
2. El Panel de Pre-Visualización (o thumbnail).
3. El Panel de Información.

3.4.1. El Panel de Diseño.

El Panel de Diseño es el módulo de la herramienta más complejo, a nivel lógico y funcional. En el mismo se representa el esquema relacional diseñado, de forma gráfica, e interactúa en multitud de maneras con el usuario.

Para la implementación de este módulo se ha hecho uso de la librería *JUNG* (Java Universal Networks/Graphs FrameWork), aunque sólo se ha requerido su aspecto visual para los fines de la herramienta.

El Panel de Diseño tiene dos funcionalidades internas que requieren especial mención. Ambas funcionalidades son la impresión en papel del esquema diseñado y la exportación a fichero gráfico del mismo. Las dos características utilizan herramientas provistas por Java para facilitar ambas tareas, por lo que no ha sido necesaria una implementación especial dedicada para dichas acciones.

Se han redefinido y sobrecargado métodos y clases que la librería ofrece con el fin de adaptar el diseño al modelo deseado. Cabe señalar la implementación de etiquetadores de nodos y aristas para la correcta visualización de la información del esquema, así como representadores gráficos para cada tipo de nodo particular.

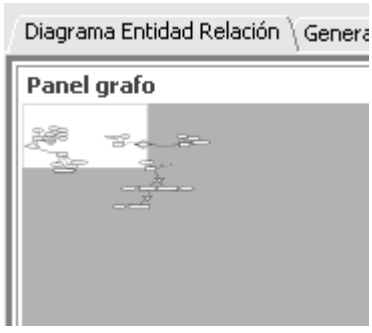
Así pues, el panel de diseño posee dos facetas destacables:

1. *Información del esquema relacional diseñado*. El Panel de Diseño posee todos los datos del esquema que se está desarrollando de forma exclusiva y los organiza en tablas dispersas (hash-tables) para un óptimo acceso a todos los datos necesarios. Estos datos son exactamente los mismos que poseen todos los demás módulos del sistema, ya que los datos almacenados son referencias a los mismos, y la única modificación que se posee es en la estructura de contenido de los mismos.
2. *Representación y captura de eventos* en el mismo por parte del usuario. El Panel de Diseño se encarga tanto del mantenimiento gráfico del esquema como del entorno de eventos que el usuario realiza sobre el mismo. Esto significa que cualquier acción sobre el grafo será capturada por el Panel de Diseño y se realizará el protocolo habitual de mensajes.
   El módulo de Panel de Diseño es un componente interno del frame principal, pero todas las acciones que el usuario realice sobre el mismo son notificadas directamente al módulo Controlador, ya que podría considerarse más una extensión del mismo que un módulo interno.
   El Panel de Diseño maneja, en esencia, tres tipos de acciones:

1. *Pulsaciones sobre el panel*. Existen dos tipos de pulsaciones posibles sobre el panel, que dependerán del botón del ratón utilizado en cada caso.
   Si se pulsa el botón izquierdo se asocia dicha acción a la petición de información sobre un elemento. Esto significa que el panel captura la acción del usuario, selecciona dónde ha pulsado, si en un nodo del esquema o un espacio vacío, y envía al Controlador dicha información para que se pueda actualizar el Panel de Información.
   Si, por el contrario, se pulsa el botón derecho, el panel capturará el evento del usuario y le mostrará, en un menú desplegable, las posibles opciones de modificación del esquema que son posibles en dicho espacio. Es decir, si la pulsación se realiza sobre un nodo se mostrará las posibles opciones de modificación sobre ese nodo, y si es sobre un espacio vacío, el panel mostrará un menú con las posibles opciones de inserción en dicho espacio vacío.
   Una vez capturado el evento de acción del usuario, y seleccionada la acción deseada por parte del mismo, se procede al funcionamiento estándar de paso de mensajes al Controlador.
2. *Movimiento de nodos visuales*. Si se pulsa con el botón izquierdo sobre el ratón y se realiza un movimiento del mismo sin la relajación del botón presionado, se realiza un movimiento sobre el nodo del esquema seleccionado. Ello significa que se capturará el nodo movido y la posición final del mismo para notificar al Controlador de dicha acción.
   Es posible la selección de varios nodos simultáneos y su movimiento en conjunto.
   El funcionamiento de esta característica está íntimamente ligada, y viene proporcionada gracias, a la implementación de la librería JUNG.
3. *Zoom sobre la vista del pane*l. Si se utiliza la rueda del ratón sobre el Panel se capturará dicha acción y se almacenará el sentido del movimiento.
   El objetivo de esta acción será la de ampliar o disminuir el zoom que afecta al Panel de Diseño hasta, en principio, un límite no establecido.
   Este comportamiento no altera ningún dato del esquema diseñado y, por consiguiente, no existe ningún intercambio de mensajes con ningún otro módulo del sistema.

3.4.2. El Panel de Pre-Visualización (o thumbnail).

Este panel no permite una modificación del esquema por parte del usuario, pero fue concebido para simplificar la búsqueda sobre el Panel de Diseño del esquema representado.



El Panel thumbnail muestra una mera representación alejada de todo el espacio en el que se puede desarrollar el esquema relacional. La representación es la misma, pues se pueden observar los mismos objetos gráficos en ambos paneles y, de hecho, comparten la misma información. Ello significa que un movimiento o cambio en el Panel de Diseño modificará el contenido del Panel de Pre-Visualización sin la necesidad de realizar ningún cambio en el mismo.

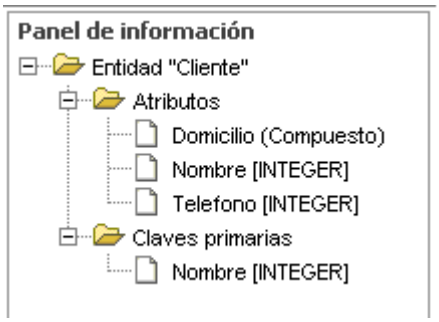
El Panel thumbnail sólo permite un tipo de acción por parte del usuario, y es desplazar la parte visible, coloreada en blanco, por su contenido. Al estar enlazados ambos paneles, la actualización de la vista en el Panel de Diseño es automática, sin la necesidad, de nuevo, de realizar ningún tipo de procedimiento extra.

Es decir, el panel de Pre-Visualización no posee ningún interfaz con el resto de módulos del sistema, ya que está vinculado a los necesarios desde su construcción.

3.4.3. El Panel de Información.

El Panel de Información muestra la información que se desee exponer de forma explícita. Ello se consigue con la pulsación del ratón sobre un nodo del esquema, por ejemplo.

La implementación del Panel de Información consta de un elemento *JTree* (o árbol gráfico) en que se muestran las características o propiedades del elemento deseado.



El interfaz que posee es, únicamente, de recepción de mensaje por parte del Controlador, y dicho mensaje ya poseerá el objeto árbol necesario para su representación.

Por la simplicidad de implementación del panel se decidió implementar dentro del módulo *frame principal*.

4. Controlador.

El controlador es el núcleo principal de la herramienta.

Se podría considerar su función realizando el símil con la estructura de un Sistema Operativo modelo cliente-servidor, también llamado micronúcleo, en el caso del kernel de dicho sistema.

El controlador tiene dos funciones principales:

1. Inicializar todos los componentes de la aplicación.
2. Controlar el paso de mensajes y coordinación de los mismos.

Todos los componentes, o módulos, de la aplicación son inicializados y mantenidos en memoria desde el arranque de la misma. La motivación de ese comportamiento viene derivada de una estabilidad general por parte de la aplicación, ya que una vez arrancada se tiene total seguridad de su fiabilidad y comportamiento.

Es el Controlador el que realiza la acción de arranque, y el módulo al que se llama desde el arranque de la herramienta y, por lo tanto, tiene la función definida de poner en marcha toda la maquinaria del sistema.

Todos los mensajes del sistema pasan por el módulo Controlador, según el esquema Vista-Controlador expuesto con anterioridad. Ello significa que el Controlador es capaz de enrutar cualquier petición entrante y realizar el desarrollo que de ella deriva.

Aunque el código de este módulo pudiera ser simplificado, o eliminado, en una implementación directa entre las demás capas, esta decisión mejora enormemente la escalabilidad del software diseñado y limita las interfaces y funcionalidades del resto de módulos de una forma tajante.

Para la selección de mensajes posibles ha sido declarada una clase enumerada TC que almacena el tipo de mensaje, tanto entrante como saliente, que puede manejar el módulo Controlador.

No se adjunta ninguna pieza de código de la clase expuesta debido a la extensa cantidad de información que maneja. Si se desea puede consultar su implementación concreta en la clase **Controlador.java**.

Por último y para ilustrar su importancia, señalar que el Controlador, en su estado final, maneja **216** tipos de mensajes.

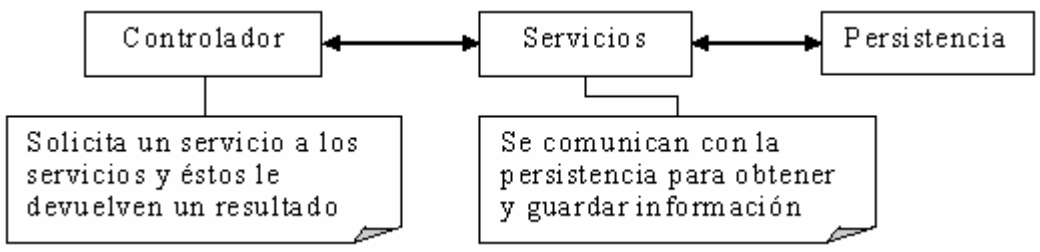
5. Servicios de aplicación.

Los servicios de aplicación se encargan de la implementación de todos los requisitos funcionales anteriormente descritos. DBDT tiene 4 tipos de servicios, agrupando en cada uno de ellos la funcionalidad de cada uno de los módulos de la aplicación.

Estos servicios son los siguientes:

1. Servicios de entidades
2. Servicios de atributos.
3. Servicios de relaciones.
4. Servicios de sistema.

Los servicios de aplicación son solicitados por el controlador y hacen uso de la capa de persistencia.



A continuación explicaremos la implementación de cada uno de los servicios.

Es importante resaltar el hecho de que tener realizada la especificación de los requisitos funcionales del sistema previamente a la implementación, ha hecho que muchos de los servicios implementados en cada uno de los módulos sean una mera traducción a JAVA de los mismos.

5.1. Servicios de Entidades.

Los servicios de entidades se centran en la implementación de los requisitos funcionales relacionados con las entidades. La implementación se encuentra en el fichero **ServiciosEntidades.java**.

A la hora de implementar los servicios de entidades hemos hecho una traducción casi literal de la especificación de los requisitos funcionales del módulo entidades. Son los siguientes:

```
public void anadirEntidad(TransferEntidad te)
public void renombrarEntidad(Vector v)
public void debilitarEntidad(TransferEntidad te)
public void anadirAtributo(Vector v)
public void eliminarEntidad(TransferEntidad te)
public void moverPosicionEntidad(TransferEntidad te)
```

Si se desea conocer la implementación concreta de estos métodos, consultar el fichero fuente o ver la especificación de los requisitos funcionales de las entidades en secciones anteriores de este documento.

Otros métodos que tienen los servicios de entidades son los siguientes:

```
public void ListaDeEntidades()
```

El método *ListaDeEntidades* accede a la persistencia mediante el DAOEntidades y devuelve al controlador una lista con todas las entidades del proyecto actual.

```
private Vector<TransferRelacion> eliminaRefererenciasAEntidad(TransferEntidad te)
```

El objetivo de este último método (privado) de la clase es la de eliminar las referencias a una entidad. Se utiliza cuando se elimina una entidad: tras la eliminación de la entidad, si había alguna referencia a ella (participaba en alguna relación) es necesario eliminar dichas referencias, ya que se producirían incoherencias en el sistema. El valor devuelto será una lista de relaciones modificadas. Si por el contrario, la entidad no está referenciada en ninguna otra relación del sistema, el vector devuelto estará vacío.

5.2. Servicios de Atributos.

Al igual que ocurre en los servicios de entidades, la implementación de los servicios de atributos es una traducción de la especificación de los requisitos funcionales de atributos.

La implementación de estos servicios se encuentra en el fichero **ServiciosAtributos.java**.

A continuación mostramos los principales métodos implementados:

```
public void anadirAtributo(Vector v)
public void eliminarAtributo (TransferAtributo ta)
public void renombrarAtributo(Vector v)
public void editarDomnioAtributo(Vector<Object> v)
public void editarCompuestoAtributo(TransferAtributo ta)
public void editarMultivaloradoAtributo(TransferAtributo ta)
public void moverPosicionAtributo(TransferAtributo ta)
public void editarClavePrimariaAtributo(Vector<Transfer> vectorDeTransfer)
```

Si se desea conocer la implementación concreta de estos métodos, consultar el fichero fuente o ver la especificación de los requisitos funcionales de los atributos en secciones anteriores de este documento.

Otros métodos implementados son los siguientes:

```
public void ListaDeAtributos()
```

Este método accede a la persistencia mediante el DAOAtributos y devuelve al controlador una lista con todos los atributos que hay en el sistema.

```
private Transfer eliminaRefererenciasAlAtributo(TransferAtributo ta)
```

El método privado *eliminaReferenciasAlAtributo* es una parte muy importante de los servicios de atributos. Es utilizado cuando se elimina un atributo: tras la eliminación de un atributo, como éste siempre se encuentra asignado a otro objeto (único) del sistema (entidad, relación u otro atributo) es necesario eliminar dichas referencias para evitar incoherencias.

Este método devuelve un objeto Transfer que contiene el único elemento del sistema que lo referencia y, por tanto, el que hay que modificar tras la eliminación del atributo en cuestión para mantener el sistema correctamente.

5.3. Servicios de Relaciones.

Al igual que en los dos casos anteriores, la implementación de los servicios de relaciones es una traducción de los requisitos funcionales. Dicha implementación se encuentra en el fichero **ServiciosRelaciones.java**.

A pesar de que a lo largo de toda esta documentación se ha hablado de dos tipos de relaciones (relaciones normales y relaciones de herencia IsA), la implementación de los servicios de relaciones engloba todos los tipos.

De acuerdo a esto, los métodos que implementan los servicios de relaciones son los siguientes:

```
public void anadirRelacion(TransferRelacion tr)
public void anadirRelacionIsA(TransferRelacion tr)
public void eliminarRelacionIsA(TransferRelacion tr)
public void eliminarRelacionNormal(TransferRelacion tr)
public void renombrarRelacion(TransferRelacion tr, String nuevoNombre)
public void debilitarRelacion(TransferRelacion tr)
public void anadirAtributo(Vector v)
public void moverPosicionRelacion(TransferRelacion tr)
public void establecerEntidadPadreEnRelacionIsA(Vector<Transfer> datos)
public void quitarEntidadPadreEnRelacionIsA(TransferRelacion tr)
public void anadirEntidadHijaEnRelacionIsA(Vector<Transfer> datos)
public void quitarEntidadHijaEnRelacionIsA(Vector<Transfer> datos)
public void anadirEntidadARelacion(Vector v)
public void editarAridadEntidad(Vector<Object> v)
public void quitarEntidadARelacion(Vector<Transfer> datos)
```

Si se desea conocer la implementación concreta de estos métodos, consultar el fichero fuente o ver la especificación de los requisitos funcionales de las relaciones en secciones anteriores de este documento.

El otro método implementado es el siguiente:

```
public void ListaDeRelaciones()
```

Este método accede a la persistencia mediante el DAORelaciones y devuelve al controlador una lista con todas las relaciones que hay en el sistema.

5.4. Servicios de Sistema.

Los servicios de sistema, tal y como se han especificado anteriormente, se centran en la validación del diseño realizado por el usuario y la generación de código en lenguaje SQL.

La implementación de estos servicios se encuentra en el fichero **ServiciosSistema.java**.

Cabe destacar que se han seguido ciertos criterios de validación, teniendo en cuenta optimizaciones del diseño y sobre todo las restricciones básicas para un buen funcionamiento de la base de datos resultante.

Los servicios del sistema, como se ha podido comprobar en la especificación, acceden a los datos del proyecto en curso a través de objetos DAO (patrón arquitectónico ya comentado en otra sección de este documento).

5.4.1. Validación del diseño.

La validación del diagrama Entidad Relación, se realiza mediante el recorrido directo de los ficheros de datos. La idea principal es ir comprobando uno a uno los objetos del modelo, comenzado por los atributos, siguiendo con las entidades y terminando con las relaciones.

Gracias a esta estructuración, los servicios de sistema devuelven secuencialmente mensajes al controlador con el resultado de cada uno de los pasos de validación.

Se han implementado tres tipos de mensajes resultantes al comprobar las características del diseño. Además estos mensajes van acompañados de un comentario informativo que especifica la situación concreta que ha tenido lugar. Son los siguientes:

- **Éxito**: Cuando el resultado del paso de validación es satisfactorio.
- **Error**: Cuando el resultado del paso de validación no cumple el criterio tratado.
- **Aviso**: Cuando el resultado del paso de validación cumple los criterios pero se detecta algún posible fallo de diseño a nivel de eficiencia o accesibilidad.

Ejemplos de estos tres tipos de mensajes son ilustrados por las siguientes imágenes:

Validando *Paciente*
Validando claves primarias de la entidad.
ERROR: La entidad no tiene clave primaria.

Al validar la entidad Paciente, el sistema detecta que la entidad no tiene definida ninguna clave primaria.

Validando *direccion*
Validando la exclusividad del atributo
ÉXITO: El atributo direccion pertenece a una sola entidad.
Validando el dominio del atributo
ÉXITO: El dominio del atributo es correcto.
Validando hijos atributo compuesto.
AVISO: El atributo solo tiene un subatributo.

El mensaje de aviso se corresponde con un atributo que está definido como compuesto y únicamente tiene un componente.

Validando *NumeroColegiado*
Validando la exclusividad del atributo
ÉXITO: El atributo NumeroColegiado pertenece a una sola entidad.
Validando el dominio del atributo
ÉXITO: El dominio del atributo es correcto.

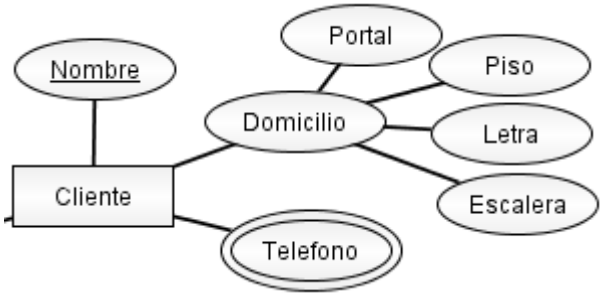
En esta imagen se observa que la validación del atributo es satisfactoria en todos los casos.

5.4.1.1. Validación de atributos.

Los atributos del diseño han de cumplir determinadas condiciones para el futuro funcionamiento del sistema. Se han implementado funciones específicas para cada una de dichas condiciones.

```
private boolean validaDominio(TransferAtributo ta)
private boolean validaFidelidadAtributo(TransferAtributo ta)
private boolean validaCompuesto(TransferAtributo ta)
```

Los posibles *roles* de un atributo en un modelo E/R son variados. Puede ser un atributo multivalorado y/o compuesto, puede pertenecer a una entidad, a una relación o ser un subatributo de otro. Todas estas posibilidades se han tenido en cuenta a la hora de comprobar la corrección del diseño.



Cabe destacar, que ciertas características que podrían considerarse propias de los atributos, como el carácter de clave de una entidad por ejemplo, son comprobadas en la fase de validación de entidades dado que se ha considerado que resulta mas cómodo y fiable debido a la estrategia seguida para el almacenamiento de los datos.

Como primer parámetro a controlar está lo que hemos denominado **fidelidad del atributo**. El objetivo de esta comprobación es asegurarse de que un atributo concreto no esta repetido en varias entidades o relaciones. Hemos de precisar que al hablar de repeticiones no nos referimos a atributos que tengan el mismo nombre, sino a una posible variación a nivel de acceso a los datos a través de los cuales se organiza el sistema.

La segunda comprobación que tiene lugar en la validación de atributos se centra en controlar el dominio de éstos. La idea básica es asegurarse de que los dominios introducidos por el usuario existen en el sistema. Además, cuando se trata un atributo compuesto, por convenio en el diseño de la aplicación se considera que el atributo padre deberá tener dominio nulo comprobándose en consecuencia los tipos de los hijos.

Precisamente sobre este tipo de atributos, los compuestos, se centra el último de los criterios de validación de esta sección.

El objetivo de estas comprobaciones es controlar el número de hijos que posee dicho atributo. Al ser compuesto, ha de tener al menos un hijo, con lo que si se detecta que esto no sucede, la validación lo considerará un error. Si el número de hijos es exactamente uno, se considerara correcto, pero el servicio de sistema enviara un mensaje de aviso indicando una situación potencialmente incómoda.

5.4.1.2. Validación de entidades.

Análogamente a los atributos, las entidades pueden cumplir ciertos roles en modelo ER. Pueden ser débiles o fuertes y esto influye en la validación de cada una de ellas.

```
private boolean validaKey(TransferEntidad te)
private boolean compruebaClaveCompuesto(Vector clavesEntidad,TransferAtributo ta)
private boolean validaNombresAtributosEntidad(TransferEntidad te)
private void validaFidelidadEntidadEnIsA(TransferEntidad te)
```

La primera validación que se realiza está centrada en las claves de la entidad. Una entidad fuerte esta obligada por diseño a tener un atributo clave. Salvo ciertas excepciones, como sería pertenecer a una relación de herencia, el sistema dará un error en el modelo E/R si no se cumple esta condición. Además en las entidades débiles se controla también la existencia de atributos discriminantes.

Surgen situaciones variadas para asociar una clave a una entidad. Por ejemplo, si se da el caso de que un atributo compuesto es clave, se consideran claves todos sus hijos. Otro caso particular es la posibilidad de haber declarado como clave un atributo multivalorado. Esto el sistema lo considera inaceptable, ya que no es un atributo que distinga inequívocamente una entidad de otra.

La segunda validación que se realiza en las entidades se trata de controlar los nombres de sus atributos. Devuelve un error si encuentra atributos con el mismo nombre dentro de una misma entidad.

Como tercera validación, comprueba la intervención de la entidad en relaciones de herencia. Se procura controlar que dicha entidad sea padre solo de una relación de herencia, pero en caso contrario, solo da un aviso dado que no constituye un error en si.

5.4.1.3. Validación de relaciones.

Sin lugar a dudas, las comprobaciones más complejas se realizan con las relaciones. Hemos tenido que controlar el papel de cada una de las entidades que intervienen dependiendo del tipo de relación implicada.

Al igual que las entidades y los atributos, las relaciones pueden actuar con distintos roles: débiles, normales o IsA (relaciones de herencia). Para cada uno de estos papeles hay un método en los servicios de sistema que evalúa su corrección.

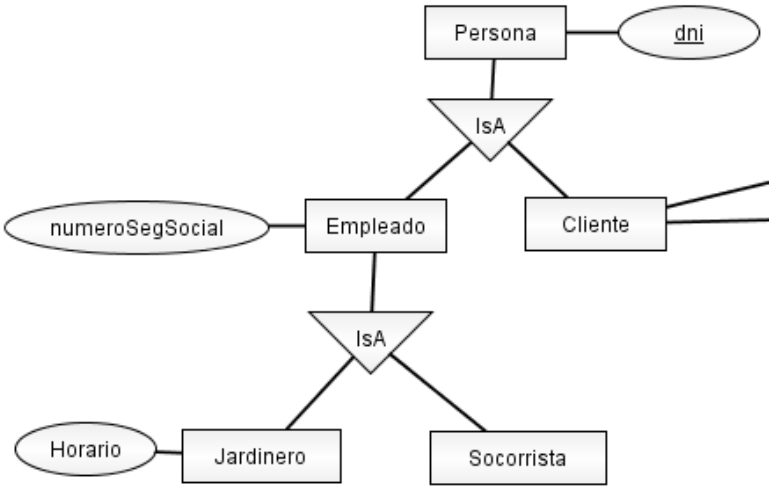
Es de destacar que (como se explica en la implementación de la capa de persistencia de datos), se recurre a una clase llamada **EntidadYAridad.java** que almacena cada grupo de entidades que participan en una relación y la aridad con la que intervienen.

```
private boolean validaComponentesRelacionIsA(TransferRelacion tr)
private boolean validaComponentesRelacionNormal(TransferRelacion tr)
private boolean validaComponentesRelacionDebil(TransferRelacion tr)
```

5.4.1.4. Validación de relaciones de herencia.

Las comprobaciones básicas que se tienen en cuenta para este tipo de relaciones están basadas en los papeles de las entidades que intervienen.

Se comprueba el numero de hijos que posee la relación y obviamente la existencia de un padre.

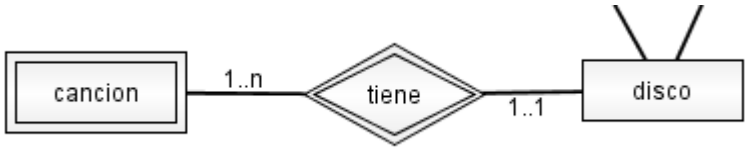


5.4.1.5. Validación relaciones normales.

Esta es la validación más sencilla. Para este tipo de relaciones, la aplicación comprueba que existan entidades implicadas y además destaca el hecho de que exista una relación auto-relacionada.

5.4.1.6. Validación de relaciones débiles.

Esta sección es la validación más peliaguda del proceso. Además de comprobar la existencia de entidades implicadas, controla el papel de estas. Concretamente, para considerar una relación como débil, debe participar en ella una entidad débil como mínimo. No obstante, dicha entidad puede participar como fuerte en otra relación diferente y eso también se comprueba recorriendo los ficheros de datos.



5.4.2. Generación de código SQL y del Modelo Relacional

Esta segunda parte de los servicios del sistema se centra en la generación del código derivado del modelo E/R diseñado y en la generación del Modelo Relacional asociado.

El objetivo es proporcionar al usuario las sentencias SQL que permiten crear las tablas que tendrá la base de datos.

Para poder desarrollar esta implementación se ha utilizado la clase **Tabla.java** que almacena los valores de las claves (primarias y foráneas), atributos etc.

5.4.2.1. La clase Tabla.java.

Cada objeto de esta clase tiene como atributos: el nombre de la tabla, la lista de atributos, la de claves primarias y la de foráneas.

```
private String nombreTabla;
private Vector<String[]> atributos;
private Vector<String[]> primaries;
private Vector<String[]> foreigns;
```

Cada una de estas listas esta implementada como un vector de arrays de String. Dentro del código estos arrays tienen un tamaño definido de 3 componentes, el nombre del atributo, su dominio y el nombre de la tabla de donde procede. Así, para la creación de tablas que requieran claves foráneas, se puede acceder a esa información fácilmente.

Ejemplo:

```
Atributos[0] = DNI;
Atributos[1] = Char(10);
Atributos[2] = Cliente;
```

Esta clase Tabla, posee además, dos métodos para obtener el código SQL.

```
public String codigoEstandarCreacionDeTabla()
public String codigoEstandarClavesDeTabla()
```

El primero de ellos genera las sentencias de creación iniciales, borrando anteriormente del sistema una posible tabla idéntica. A continuación, produce una sentencia CREATE acompañada del nombre de la tabla y sus atributos con sus dominios correspondientes.

El segundo método, comprueba las tablas y realiza las sentencias que permiten configurar las claves de cada una. Gracias a esto se pueden conectar unas tablas con otras plasmando el diseño del modelo E/R.

Análogamente, para obtener el Modelo Relacional se ha implementado otro método que utiliza la estructura de la tabla.

```
public String modeloRelacionalDeTabla()
```

Como último apunte de esta clase se puede destacar que detecta y solventa las situaciones con tablas con nombres de atributos comunes y la corrección de nombres con guiones medios y espacios en blanco. Este último detalle es bastante importante debido a que los clientes SQL más habituales no aceptan este tipo de escritura.

5.4.3. Implementación específica.

Una vez explicada la clase tabla, podemos comentar la implementación de la sección de los servicios de sistema.

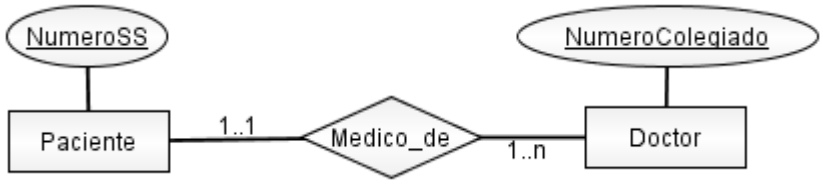
Los servicios poseen tres tablas hash que almacenan cada objeto tabla resultante. Como clave de cada una de estas tablas hash se utiliza el identificador exclusivo en el sistema del elemento implicado.

```
private Hashtable<Integer,Tabla> tablasEntidades;
private Hashtable<Integer,Tabla> tablasRelaciones;
private Hashtable<Integer,Tabla> tablasMultivalorados;
```

El sistema genera las tablas correspondientes a cada una de las entidades y relaciones. Para las primeras resulta algo menos complejo, dado que la mayor preocupación es controlar las entidades débiles y sus claves foráneas.

En cambio, a la hora de generar las tablas de una relación entran en juego muchos factores. Primero hay que apreciar de qué tipo de relación se trata, ya que las débiles no generan tablas según los convenios que hemos alcanzado. Cuando tenemos relaciones de herencia tenemos diversos casos particulares para éstas. Como se explica en la teoría sobre las bases de datos que se incluye al principio de esta documentación, se generan tablas tanto para el padre como para los hijos de la dicha relación. No obstante, se controla la asignación de claves ya que éstas se heredan.

Para las relaciones normales, el principal factor que se tiene en cuenta es la aridad de las entidades implicadas. Cuando se tiene un diseño como el del ejemplo siguiente, en el que la entidad *paciente* participa con una cardinalidad (1..1) y la entidad doctor con aridad (1..n) se entiende que la clave de la relación *medico_de* será la clave de doctor.



Este tipo de situaciones están todas contempladas a la hora de crear las tablas que nos ayudaran a obtener tanto el código SQL como el Modelo Relacional.

A continuación se muestra el resultado de la generación de código del ejemplo anterior.

SECCIÓN DE CREACIÓN DE TABLAS

```
DROP TABLE Doctor;
CREATE TABLE Doctor(NumeroColegiado INTEGER);

DROP TABLE Paciente;
CREATE TABLE Paciente(numeroSS INTEGER);

DROP TABLE Medico_De;
CREATE TABLE Medico_De(NumeroColegiado INTEGER, numeroSS INTEGER);
```

SECCIÓN DE ESTABLECIMIENTO DE CLAVES

```
ALTER TABLE Doctor ADD PRIMARY KEY (NumeroColegiado);
ALTER TABLE Paciente ADD PRIMARY KEY (numeroSS);
ALTER TABLE Medico_De ADD FOREIGN KEY (NumeroColegiado) REFERENCES Doctor (NumeroColegiado);
```

La última característica resaltable de la implementación de estos servicios tiene que ver con la obtención de un fichero .sql que el usuario puede obtener fácilmente para cargarlo en cualquier cliente SQL.

6. Persistencia

La persistencia de los datos en DBDT esta basada en el patrón DAO con acceso a ficheros XML. En esta sección explicaremos cómo se gestionan esos datos y cual es su estructura concreta en este fichero.

Además, en esta aplicación existe un identificador numérico exclusivo para cada una de las entidades, relaciones o atributos residentes en el sistema, garantizando así el acceso inequívoco éstos.

Para mas información sobre el funcionamiento del patrón DAO, se puede consultar la sección de esta documentación dedicada a los patrones utilizados.

6.1. XML (Extensible Markup Language).

XML es un metalenguaje estructurado que permite almacenar información de manera organizada. Esta información puede ser fácilmente compartida con cualquier software siempre que se tenga un analizador adecuado.

La estructura de un fichero XML es sencilla y jerárquica. Esta basada en etiquetas, similares a HTML, que se encuentran anidadas con el contenido correspondiente dentro de ellas. Para que un documento este bien formado y no de lugar a errores, todas las etiquetas deben de estar cerradas y además debe de existir un único elemento raíz del que todos los demás serán parte.

Es posible configurar la codificación de los datos que se van a almacenar para adaptarlos a distintos alfabetos o lenguajes.

A continuación, mostraremos como son los ficheros XML con los que trabaja DBDT.

6.2. Ficheros XML en DBDT

El almacenamiento de la información en esta aplicación esta organizado mediante tres ficheros: *entidades.xml*, *atributos.xml* y *relaciones.xml*. Todos los documentos están codificados con el estándar ISO-8859-1. Otra característica común es la inclusión de un campo *posición* que sirve para almacenar la posición del objeto en el panel de diseño.

6.2.1. Entidades.xml

Es una lista de entidades cuyo atributo es el próximo identificador a asignar a la siguiente entidad que ingrese en el sistema (esta parte es análoga en los ficheros de relaciones y atributos).

Como campos de cada entidad almacenamos la información básica:

- **Nombre**: El nombre representativo de la entidad.
- **Carácter de débil**: Indica si la entidad es débil o no.
- **Lista de atributos**: Una lista de identificadores. Cada uno indica inequívocamente a un atributo del fichero *atributos.xml*.
- **Lista de claves primarias**: Una lista de identificadores. Cada uno indica inequívocamente a un atributo del fichero *atributos.xml* que además tiene carácter de clave dentro de la entidad tratada.

A continuación mostramos un extracto de un posible fichero *Entidades.xml*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ListaEntidades proximoID="46">
    <Entidad idEntidad="6">
        <Nombre>Persona</Nombre>
        <Debil>false</Debil>
        <ListaAtributos>
            <Atributo>41</Atributo>
        </ListaAtributos>
        <ListaClavesPrimarias>
            <ClavePrimaria>41</ClavePrimaria>
        </ListaClavesPrimarias>
        <Posicion>389,210</Posicion>
    </Entidad>
```

6.2.2. Atributos.xml

Es una lista de atributos. El próximo identificador a asignar esta almacenado en dicha lista y los campos de cada atributo son los siguientes:

- **Nombre**: El nombre representativo del atributo.
- **Dominio**: El dominio del atributo
- **Carácter de compuesto**: Un valor booleano que indica si es un atributo compuesto o no.
- **Lista de Componentes**: En caso de que sea un atributo compuesto, aquí se almacenan los identificadores de los subatributos que tiene asociados.
- **Multivalorado**: Un valor booleano que indica si el atributo es multivalorado o no.

A continuación mostramos un extracto de un posible fichero *Atributos.xml*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ListaAtributos proximoID="70">
    <Atributo idAtributo="35">
        <Nombre>tasa-credito</Nombre>
        <Dominio>INTEGER</Dominio>
        <Compuesto>false</Compuesto>
        <ListaComponentes/>
        <Multivalorado>false</Multivalorado>
        <Posicion>658,152</Posicion>
    </Atributo>
    <Atributo idAtributo="36">
        <Nombre>numeroSegSocial</Nombre>
        <Dominio>INTEGER</Dominio>
        <Compuesto>false</Compuesto>
        <ListaComponentes/>
        <Multivalorado>false</Multivalorado>
        <Posicion>185,99</Posicion>
    </Atributo>
    <Atributo idAtributo="37">
        <Nombre>Horario</Nombre>
        <Dominio>INTEGER</Dominio>
        <Compuesto>false</Compuesto>
```

6.2.3. Relaciones.xml

Es una lista de relaciones. Análogamente a las entidades y los atributos, lleva el control sobre el próximo identificador de relación a aplicar. Los campos son los siguientes:

- **Nombre**: El nombre representativo de la relación.
- **Tipo**: Almacena el tipo de relación del que se trata: débil, normal o de herencia (IsA).
- **Lista de entidades y aridades**: Almacena una lista con los identificadores de cada una de las entidades participantes en la relación y la aridad con la que se implican. Más adelante se mostrara el uso de la clase *entidadYAridad.java.*
- **Lista de atributos**: Una lista de identificadores de atributos.

A continuación mostramos un extracto de un fichero *Relaciones.xml*.

```
            <ListaEntidadesYAridades>
                <EntidadYAridad>(42,1,n)</EntidadYAridad>
                <EntidadYAridad>(43,1,1)</EntidadYAridad>
            </ListaEntidadesYAridades>
            <ListaAtributos/>
            <Posicion>759,58</Posicion>
        </Relacion>
        <Relacion idRelacion="15">
            <Nombre>Medico_de</Nombre>
            <Tipo>Normal</Tipo>
            <ListaEntidadesYAridades>
                <EntidadYAridad>(44,1,1)</EntidadYAridad>
                <EntidadYAridad>(45,1,n)</EntidadYAridad>
            </ListaEntidadesYAridades>
            <ListaAtributos/>
            <Posicion>354,278</Posicion>
        </Relacion>
    </ListaRelaciones>
```

6.3. Implementación de los DAOs

Como sabemos, el patrón DAO nos propone implementar métodos de acceso y modificación a los datos. En el caso de esta aplicación, estos datos están organizados, como comentábamos en el apartado anterior, en ficheros XML.

Existe una clase DAO para cada uno de los ficheros del sistema y cada una de estas clases tiene cinco métodos comunes: **añadir**, **consultar**, **modificar**, **borrar** y **listar**.

Además existe una clase llamada *entidadesYAridades.java* que ayuda a organizar las entidades implicadas en una relación.

Para poder utilizar la información de los archivos XML, cada uno de los DAOs posee dos métodos de apertura y cierre del documento. Estos métodos recurren a la clase *DocumentBuilderFactory* que proporciona un parser para el procesamiento del fichero.

```
private Document dameDoc()
private void guardaDoc() throws IOException
```

La implementación de cada uno de estos métodos es la siguiente:

```
private Document dameDoc() {
    Document doc = null;
    DocumentBuilder parser = null;
    try {
        DocumentBuilderFactory factoria = DocumentBuilderFactory.newInstance();
        parser = factoria.newDocumentBuilder();
        doc = parser.parse(this.path);
    }
    catch (Exception e) {
        JOptionPane.showMessageDialog(
                null,
                "ERROR:\n" +
                "Error inesperado en el fichero \"atributos.xml\"",
                "DBDT",
                JOptionPane.ERROR_MESSAGE);
    }
    return doc;
}
private void guardaDoc() throws IOException {
    OutputFormat formato = new OutputFormat(doc, "ISO-8859-1", true);
    StringWriter s = new StringWriter();
    XMLSerializer ser = new XMLSerializer(s, formato);
    ser.serialize(doc);
    // El FileWriter necesita espacios en la ruta
    this.path = this.path.replace("%20"," ");
    FileWriter f = new FileWriter(this.path);
    this.path = this.path.replace(" ", "%20");
    ser = new XMLSerializer(f, formato);
    ser.serialize(doc);
}
```

Después de obtener la variable *doc*, que nos permite la lectura y escritura del fichero XML, recurrimos a la clase *Node*, que nos proporciona cada uno de los elementos que se obtienen al procesar el documento. Lógicamente podemos obtener listas de nodos con la clase *NodeList*.

Existen una amplia gama de sentencias para la modificación, eliminación e inserción de información. Cabe destacar que toda la información que la aplicación introduce en los XML esta en formato texto y al recuperarla se realiza el casting adecuado para cada caso.

Todos los métodos principales de los DAOs, devuelven o requieren un objeto Transfer, exceptuando la función de listado que devuelve una lista de Transfer.

6.4. La clase EntidadYAridad.java

Esta clase sirve como ayuda auxiliar para almacenar la información sobre la participación de las entidades en las relaciones. Posee tres atributos: El identificador de la entidad, el principio del rango de la aridad y el final de éste.

```
int entidad;
int pRango;
int fRango;
```

Además los objetos de esta clase poseen métodos que permiten la interpretación de su información desde los ficheros XML. Así, cuando se introducen rangos totales (por ejemplo 1..n), el sistema es capaz de asimilarlos y almacenarlos correctamente.

7. Librerías externas utilizadas.

Para la implementación de la herramienta se ha hecho uso de librerías externas, o no declaradas y definidas por el usuario. Dichas librerías simplifican, en la medida de lo posible, una implementación partiendo de cero y han hecho posible la mejora de componentes inherentemente internos a la aplicación diseñada.

Las librerías utilizadas son:

- **JRE System Library (v. 1.6.0).** Es la librería que proporciona la máquina virtual de *Java* y contiene estructuras de datos, optimizadas en muy alto grado, que resultan útiles en la implementación de código.
  Su utilización es libre y, por lo tanto, permite la distribución de cualquier proyecto derivado de su uso de forma no privativa.
- **Collections Generic (v. 4.01)**. Esta librería es utilizada para definir transformaciones internas en el diseño de los esquemas relacionales. Su uso es derivado de la librería *JUNG*, que requiere dicha funcionalidad.
  La licencia que posee es *Apache License Version 2.0* y, por consiguiente, es de libre distribución.
- **Colt (v. 1.2.0).** Esta librería es utilizada de forma interna por la librería principal JUNG y no ha sido necesaria referencia alguna desde la implementación propia.
  La licencia que posee es *The Artistic License* y, al igual que la librería anteriormente expuesta, es de libre distribución.
- **Concurrent (v. 1.3.4)**. Esta librería es utilizada de forma interna por la librería principal *JUNG* y no ha sido necesaria referencia alguna desde la implementación propia.
  Esta librería no posee licencia explícita, pero su creador, *Doug Lea*, permite su libre uso en la página Web oficial del proyecto:
  http://g.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html
- **JUNG - Java Universal Network/Graph Framework (v. 2.0).** Librería utilizada para la representación gráfica de los esquemas relacionales.
  Ha sido necesaria una reimplementación de mucha funcionalidad ya introducida en la librería, pero cabe destacar su completitud tanto en el tratamiento de grafos como en su representación visual.
  La licencia que posee es *Berkeley Software Distribution (BSD) license* y permite una libre distribución.
- **Looks (v. 2.1.4)**. Looks es parte de la librería gráfica *Jigloo*. Esta librería visual mejora la claridad del contenido y el acabado gráfico de la aplicación, además de su diseño. El paquete *Looks* se ha utilizado para mejorar los efectos y objetos gráficos ofrecidos por Swing, la librería gráfica de Java.
  La licencia de distribución que posee es *CloudGarden.com SOFTWARE USER AGREEMENT*, la cual permite su distribución, incluso en instituciones académicas, si los desarrolladores no han sido contratados por las mismas.
  Enlace oficial del proyecto:
  http://www.cloudgarden.com/jigloo/

Se ha intentado, en todo momento, el uso de librerías de libre acceso y libre distribución para eliminar los posibles límites que ello podría acarrear.

La licencia bajo la que la herramienta se acoge, por lo tanto, será acorde a la más restrictiva de las expuestas previamente.

Manual de usuario.

1. Introducción.

Este manual pretende explicar las nociones básicas de interacción entre el usuario y la aplicación *DataBase Design Tool*.

El objetivo es que el usuario comprenda la usabilidad del programa de una forma sencilla y sepa afrontar las tareas básicas que se pueden desempeñar en el mismo.

Una cuestión, que ha de remarcarse desde un principio, es que DataBase Design Tool **no posee botón de guardado** o menú a tal efecto. La implementación de la herramienta trabaja de forma persistente sobre los ficheros del espacio de trabajo, y ello significa que en todo momento se trabaja con un proyecto coherente y a prueba de cierres inesperados.

Toda la interacción entre el usuario y la aplicación se realiza de forma primaria con el ratón, y de forma secundaria con el teclado. El uso del ratón es el estándar en la mayoría de aplicaciones, con uso tanto de los dos botones primarios como de la rueda que el mismo posee, y el uso del teclado se limita a la solicitud de datos que lo requieren de forma obligatoria, como la selección del nombre de una Entidad, para así simplificar la usabilidad y facilidad de la aplicación.

A lo largo de la explicación de las diversas opciones de la herramienta se adjuntarán imágenes explicativas del funcionamiento de las mismas, para la comodidad del lector.

En secciones posteriores existe un glosario con opciones y decisiones gráficas de diseño útiles para el usuario.

2. Requisitos del sistema.

El único requisito que solicita Database Design Tool es la instalación de la máquina virtual de *Java* de *Sun Microsystems (JRE v. 1.6.0* o posterior*)*.

No se ha verificado la funcionalidad de la elección de otra máquina virtual ni de compilaciones de la aplicación en código binario.

La herramienta trabaja de forma correcta, y ha sido verificada, en entornos *Windows XP* y distribuciones *GNU/Linux* (*Ubuntu* y *Debian*).

3. Instalación y ejecución de la aplicación.

Dado el objetivo de realizar una aplicación multiplataforma, utilizable en distintos sistemas, no se ha realizado un instalador de la misma para ninguna arquitectura específica.

La aplicación se proporciona de forma auto-ejecutable contenida en un fichero *JAR* (*Java ARchives*). La instalación, por tanto, se completaría con la descarga de dicho fichero y su localización en un lugar deseado a elección del usuario.

La ejecución de la aplicación es automática, siempre que la extensión de ficheros *JAR* esté asociada a la máquina virtual de *Java* en la máquina destino.

De no ser así probablemente se abra con una aplicación de tratamiento de ficheros comprimidos, pues es lo que es además de ser un fichero ejecutable. Habrá de asociarse la

extensión *JAR* entonces a la aplicación *JRE Java Runtime* y se procederá a la ejecución de la misma realizando doble *click* sobre el fichero presentado.

4. Glosario

Hay que destacar que el uso de la rueda del ratón dentro de cualquier panel de representación del diseño modificará el zoom sobre el mismo. Esta cualidad resulta útil para esquemas relativamente grandes donde se desea tener una visión más amplia que la que ofrece el interfaz de partida.

A continuación se presenta la representación gráfica de los distintos nodos posibles en el interfaz de la aplicación:

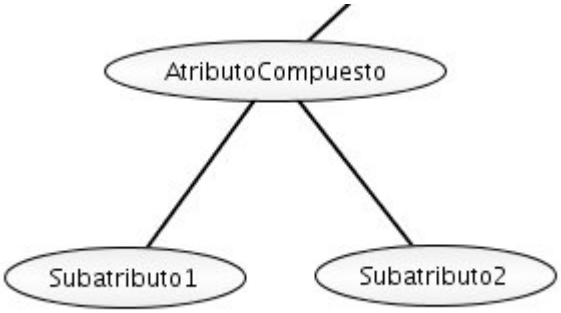
Entidad fuerte

Entidad débil

Atributo simple

Atributo clave primaria

Atributo compuesto

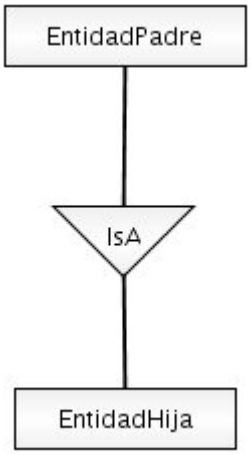


Atributo multivalorado

Relación fuerte

Relación débil

Relación IsA



5. Empezando a usar DBDT

5.1. Crear un nuevo proyecto

Se puede crear un nuevo proyecto de dos formas distintas. Ambas poseen el mismo interfaz y la misma secuencia de pasos, pero dependen del lugar en el que se intente generar el nuevo espacio de trabajo.

DBDT trabaja con espacios de trabajo. Un espacio de trabajo, o *Workspace*, es una carpeta dedicada exclusivamente al almacenamiento de la información de un proyecto.

Se permiten espacios de trabajo compartidos con otros ficheros de otra índole del usuario, pero no se recomienda en ningún caso por posibles coincidencias de uso de ficheros.

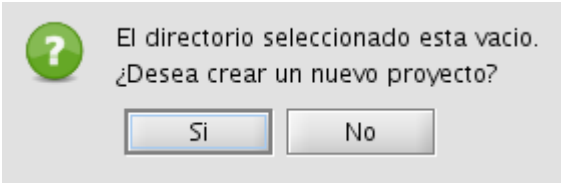
La creación de un nuevo proyecto puede ser generada a partir de dos procesos distintos.

1. Selección de un directorio vacío como espacio de trabajo en el inicio de la aplicación.
2. Selección de un directorio vacío en el proceso de Selección de espacio de trabajo dentro de la aplicación.

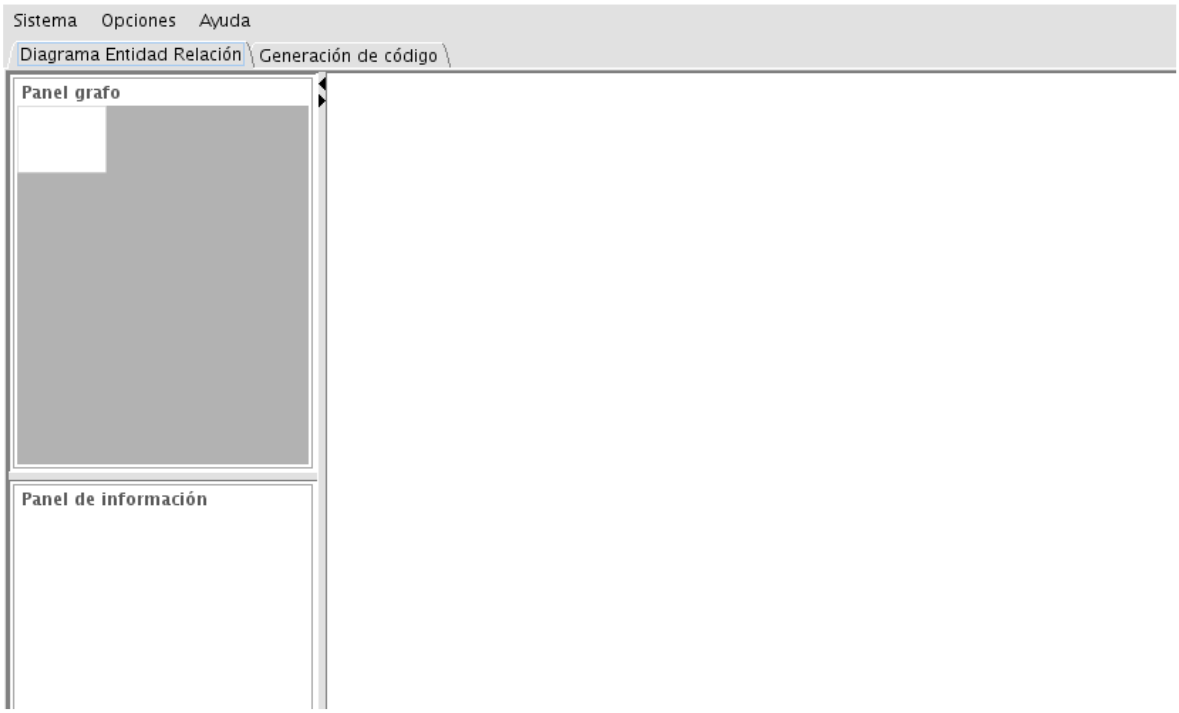
En ambas opciones se presentará inicialmente esta ventana.

En dicho diálogo se habrá de seleccionar el directorio que será considerado el espacio de trabajo. Por ejemplo, de la siguiente forma:

Una vez seleccionado el espacio de trabajo se pulsa el botón Seleccionar, y se acepta la siguiente cuestión:



Y una vez aceptada la creación del nuevo proyecto aparecerá la interfaz principal de la aplicación, con un esquema vacío:

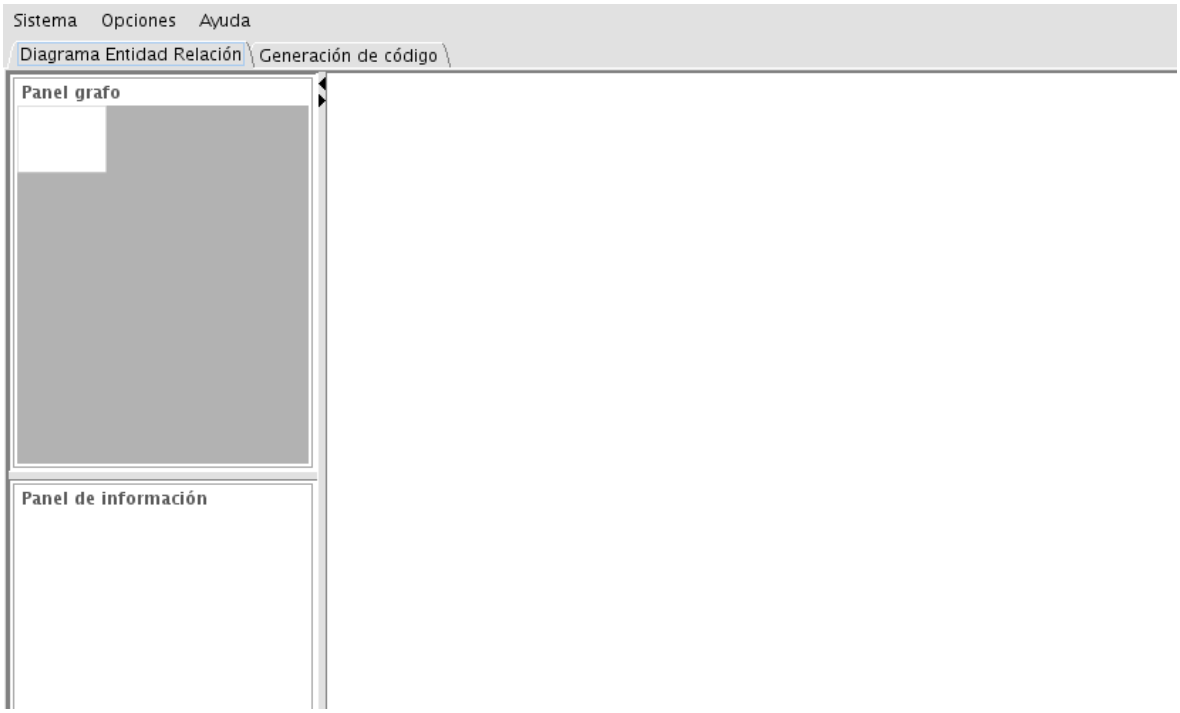


5.2. Abrir un proyecto existente.

La apertura de un proyecto existente es similar al proceso de creación de un nuevo espacio de trabajo.

La única diferencia radica en que *DataBase Design Tool* reconocerá que la carpeta seleccionada ya contiene un proyecto válido y dará paso a su carga.

Como el espacio de trabajo *XML*, en este caso, ya ha sido generado, al pulsar el botón Seleccionar se procederá a la apertura del proyecto.



5.3. Añadir entidades, atributos y relaciones a un proyecto

La adición de entidades, atributos y relaciones, que a partir de ahora se denominarán nodos, se realiza a través de la interfaz principal dentro de la ficha *Diagrama Entidad Relación*.

En dicha interfaz se pueden observar cuatro partes diferenciadas.

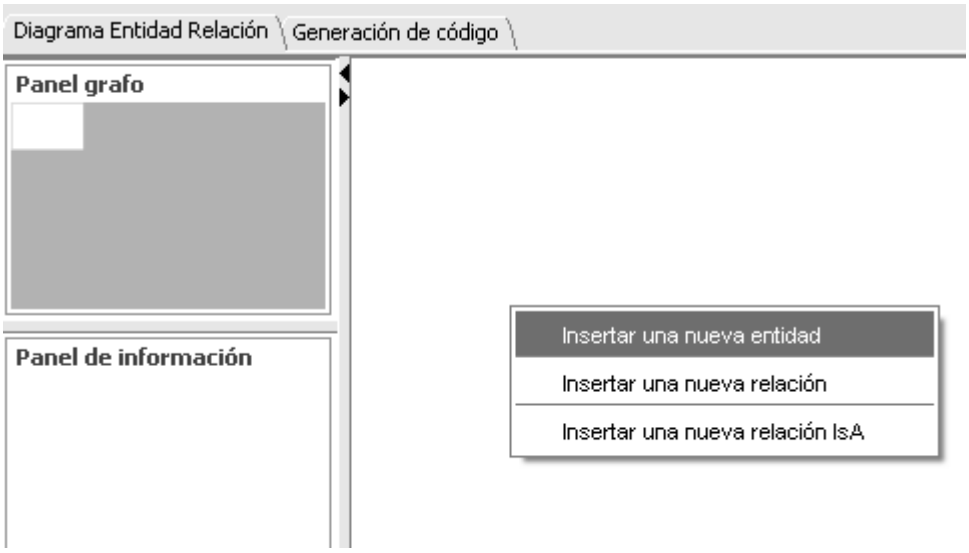
- **Panel de diseño**. Es el panel más grande que se puede apreciar en las imágenes de la aplicación. En él se representará el esquema diseñado y se interactuará con el usuario de una forma completa.
- **Panel grafo**. Este panel muestra una vista alejada del esquema, necesaria en el caso que dicho esquema sobrepase los límites visibles del panel. Permite el movimiento de la vista seleccionada en el momento actual mediante el movimiento del cuadrado interior blanco.
- **Panel de Información**. Este panel muestra la información de un nodo seleccionado mediante una pulsación de ratón. Para cada tipo de nodo se representa su información característica.
- **Panel de sucesos**. Este panel muestra la sucesión de éxitos o errores de acción por parte del usuario. Si se produce un error en un módulo interno de la aplicación, y no se ha podido realizar una tarea solicitada por el usuario, se mostrará el mensaje de error en forma de texto.

El proceso de aprendizaje constará de la generación de una entidad, que posee un atributo, la generación de otra entidad y la relación existente entre ambas.

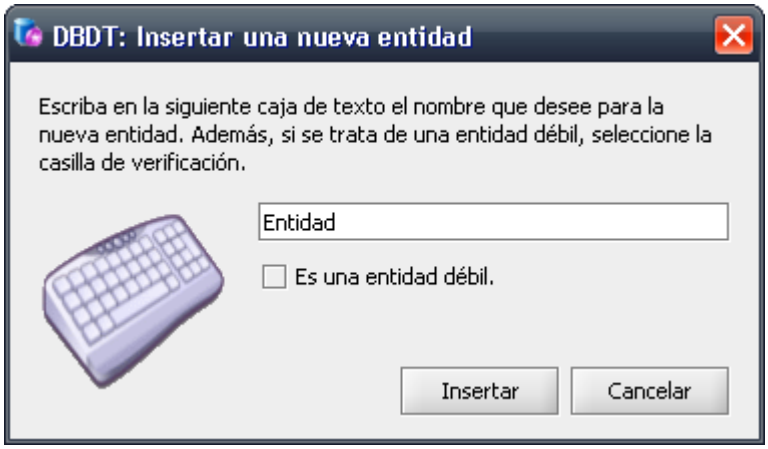
Añadir una entidad

Ha de pulsarse, dentro del panel de diseño, con el botón derecho del ratón sobre el lugar donde se desea añadir la nueva entidad.

Se expandirá el siguiente cuadro de selección.



Se selecciona la opción de *Insertar una nueva entidad* y aparecerá la siguiente ventana:



En la misma se completa el nombre de la entidad dentro del campo de texto, y se selecciona el *checkbox* si se desea que dicha entidad sea de carácter débil o no.

Si se acepta dicha inserción se obtendrá un esquema similar al siguiente.

Vemos la correcta inserción de la nueva *Entidad* tanto en el Panel de Sucesos, como en el Panel de Diseño.

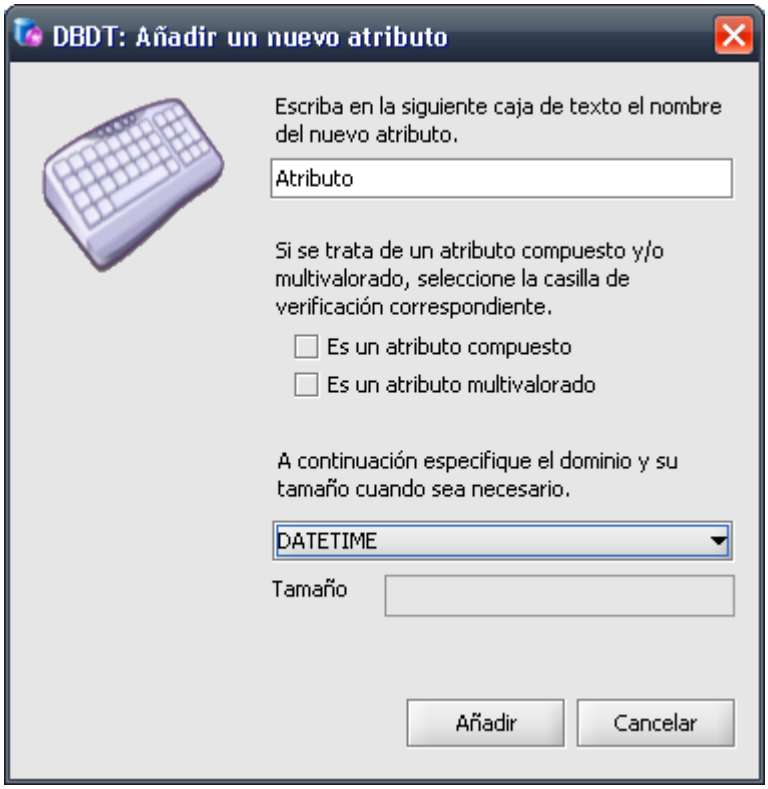
Añadir un atributo

Para añadir un atributo ha de tenerse, previamente, una entidad o una relación sobre la que relacionar dicho atributo.

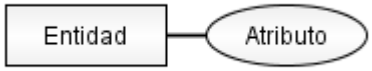
Para añadir un atributo basta con pulsar con el botón derecho del ratón dentro de una entidad, por ejemplo, y aparecerá el siguiente cuadro:

Se selecciona, dentro del mismo, *Añadir un nuevo atributo* y aparecerá la interfaz de la siguiente página.

En ella se completa, en este orden, el nombre del atributo a generar, se marcan los *checkboxes* si el atributo es compuesto o multivalorado y se especifica el tipo del atributo generado. Si el tipo requiere la especificación de un tamaño se activará el campo de texto, para rellenar el tamaño, en número, deseado para tal atributo.



Se pulsa *Añadir* y se obtiene el siguiente esquema.



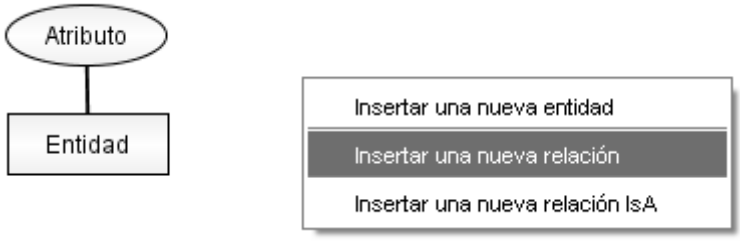
Vemos el mensaje de información de la correcta inserción del atributo en el esquema y su colocación en el diagrama.

Podemos mover ambos nodos a la posición del esquema que deseemos sin más que pulsar en un nodo y arrastrarlo al lugar deseado.

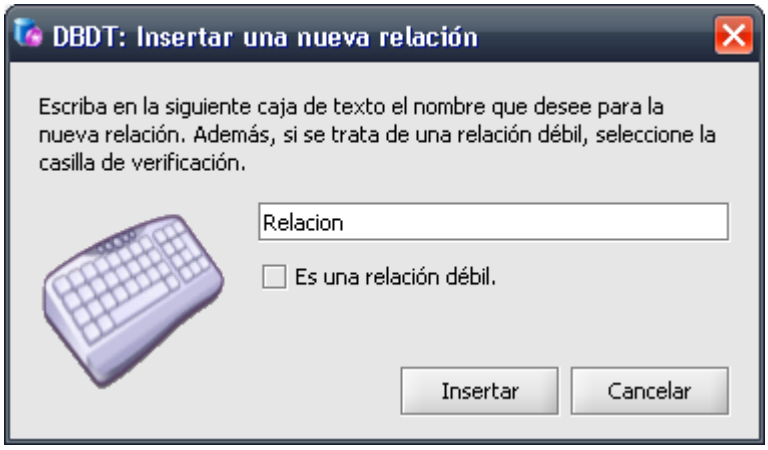
Añadir una relación

Ahora añadiremos una relación, que posteriormente relacionaremos con diversas entidades.

Para añadir una relación al sistema se pulsa con el botón derecho en un espacio libre del esquema y aparecerá el siguiente cuadro de selección.

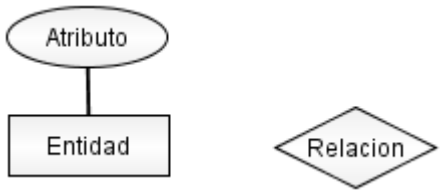


Seleccionamos *Insertar una nueva relación* y aparecerá la siguiente interfaz:



En ella hay que completar el nombre de la relación a generar y marcar el *checkbox* en caso de que se desee agregar una relación débil.

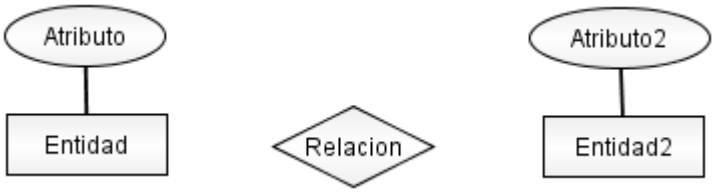
Por último, obtenemos el diseño final pulsando el botón *Insertar*.



5.4. Editar entidades, atributos y relaciones de un proyecto

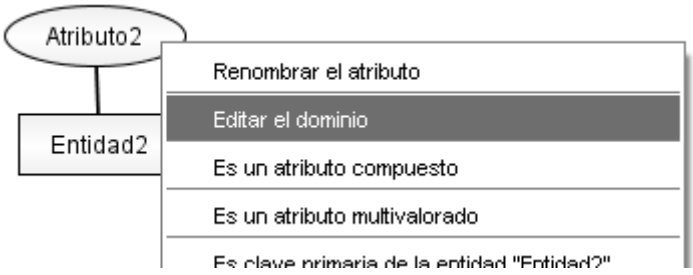
A continuación se van a realizar modificaciones en el esquema actual. No se realizarán todas las posibles, pero sí las más relevantes, aunque todas compartan el mismo esquema de realización.

Primeramente, siguiendo los pasos anteriormente descritos, generar una nueva entidad denominada *Entidad2* con un nuevo atributo denominado *Atributo2*. El esquema final quedaría como sigue.

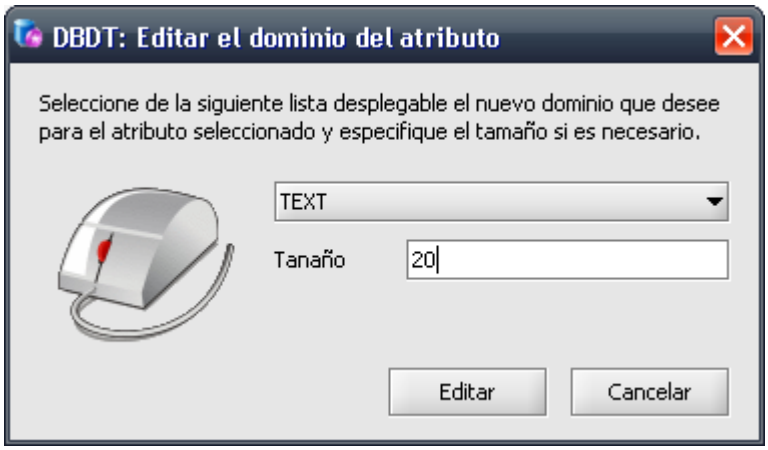


A continuación se modificará el dominio del *Atributo2*.

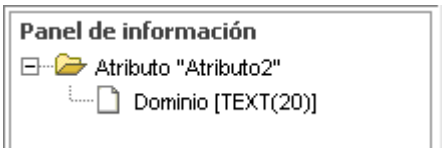
Pulsar con el botón derecho sobre el nodo *Atributo2* y seleccionar *Editar el dominio*.



Dentro del interfaz seleccionar el nuevo dominio y pulsar *Editar*.

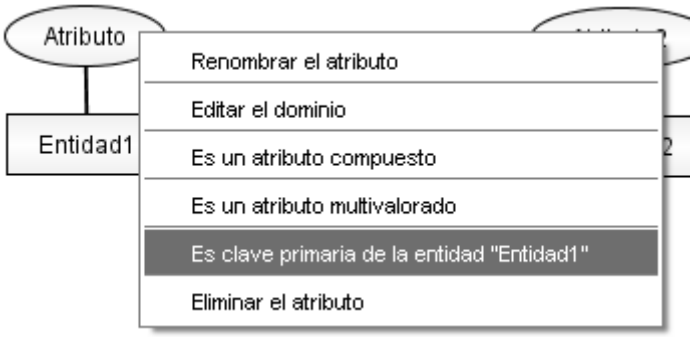


Para comprobar el cambio de dominio basta con pulsar sobre el nodo *Atributo2* y visualizar el *Panel de Información*.



Para el correcto diseño del esquema, y dado que ambas entidades son fuertes, han de poseer un atributo declarado como clave primaria. Se mostrará el proceso con el *Atributo1*.

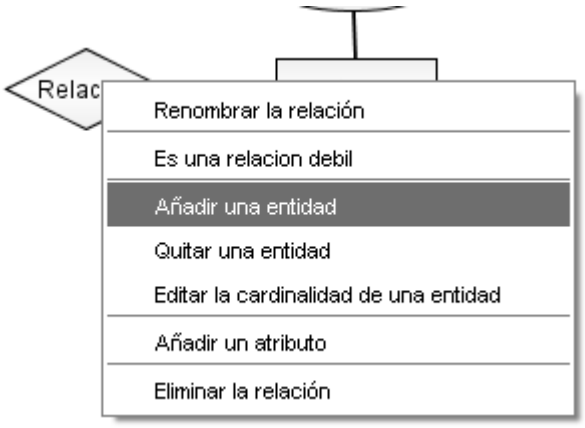
Pulsar sobre el nodo *Atributo1* y seleccionar *Es clave primaria de la entidad "Entidad1"*.



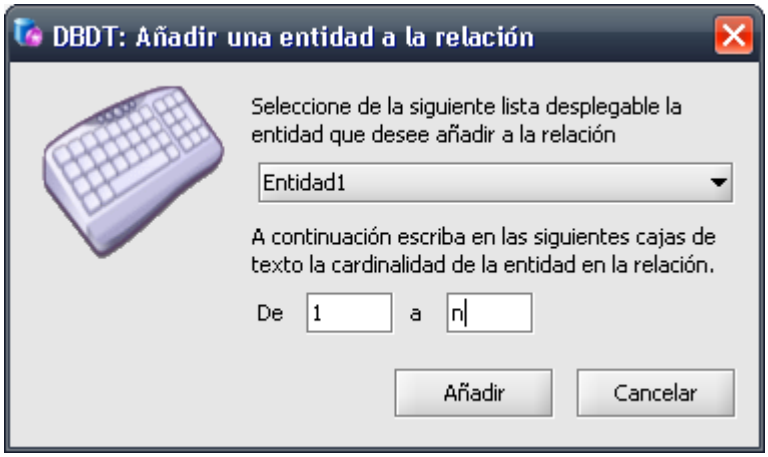
Una vez seleccionada la opción se actualizará la interfaz subrayando el nombre del nodo *Atributo1* de la siguiente forma:

El proceso finaliza con la edición de la relación, que relacionará ambas entidades. Se explicará el proceso para una entidad pues es idéntico en el segundo caso.

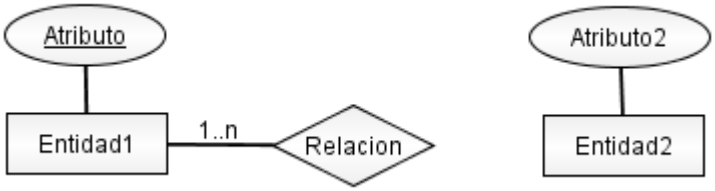
Pulsar con el botón derecho sobre la relación y seleccionar, en el cuadro de opciones, *Añadir una entidad*.



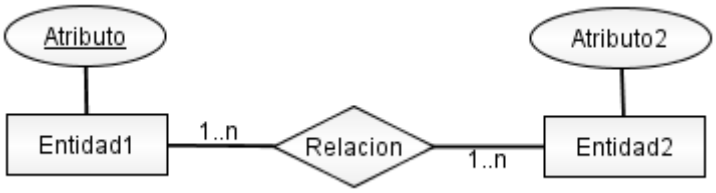
En la interfaz seleccionar dentro del objeto *combo* la entidad que se quiere relacionar y completar en los campos de texto la aridad de dicha relación. Una vez terminado pulsar *Añadir*.



El esquema quedará como se muestra a continuación:



Repetir el proceso con la *Entidad2*. El esquema final es el siguiente:



5.5. Validar diseño del diagrama E/R

Para validar el diseño generado en la etapa previa hay que situar la ventana principal en su ficha *Generación de código*.

Ahí se ven cinco botones situados en la franja izquierda de la ventana. Nombrados en orden descendente son los siguientes:

- Limpiar el área de texto de generación.
- Validación del diseño del esquema relacional.
- Generación del modelo relacional.
- Generación del script SQL en modo texto.
- Exportación del script SQL a fichero de texto.

Así pues, como se tiene un esquema ya diseñado, se procede pulsando el botón de *Validación del diseño*.

Obtenemos el siguiente resultado:

Validando *Entidad2*
Validando claves primarias de la entidad.
ERROR: La entidad no tiene clave primaria.

ERROR EN EL PROCESO. VALIDACIÓN INTERRUMPIDA

Se nos indica que el diseño no es correcto y se nos dice la razón.

Solucionamos el problema estableciendo como clave primaria de la entidad *Entidad2* el atributo *Atributo2*.

Validamos de nuevo.

VALIDACIÓN REALIZADA CON ÉXITO

Ahora ya si tenemos un diseño correcto.

5.6. Generación del Modelo Relacional.

La generación del modelo relacional se lanza con la pulsación sobre el tercer botón de la margen izquierda de la ventana. Si se desea, se puede pulsar de forma previa el primer botón para limpiar el área de texto de generación.

El resultado de la ejecución del modelo relacional es el siguiente.

MODELO RELACIONAL GENERADO:

Entidad2 = (Atributo2)

Entidad1 = (Atributo)

Relacion = (Atributo, Atributo2)

5.7. Generación del código SQL

La generación del script SQL se lanza con la pulsación sobre el cuarto botón de la margen izquierda de la ventana. Como en el caso anterior, si se desea se puede limpiar el contenido del área de texto de generación para una más fácil comprensión y legibilidad.

El código SQL generado para el esquema anterior es el siguiente:

CÓDIGO SQL DEL DISEÑO

SECCIÓN DE CREACIÓN DE TABLAS

```
DROP TABLE Entidad2;
CREATE TABLE Entidad2(Atributo2 TEXT(20));

DROP TABLE Entidad1;
CREATE TABLE Entidad1(Atributo DATETIME);

DROP TABLE Relacion;
CREATE TABLE Relacion(Atributo DATETIME, Atributo2 TEXT(20));
```

SECCIÓN DE ESTABLECIMIENTO DE CLAVES

```
ALTER TABLE Entidad2 ADD PRIMARY KEY (Atributo2);
ALTER TABLE Entidad1 ADD PRIMARY KEY (Atributo);
ALTER TABLE Relacion ADD FOREIGN KEY (Atributo) REFERENCES Entidad1 (Atributo);

ALTER TABLE Relacion ADD FOREIGN KEY (Atributo2) REFERENCES Entidad2 (Atributo2);
```

Como requisito fundamental se ha añadido el botón de ***exportación a fichero de texto plano***. Si una vez generado el código SQL, se pulsa sobre el mismo, se solicitará la ubicación y el nombre del fichero con el script generado.

6. Ejemplo completo de uso.

Base de Datos para un colegio.

Imaginemos que queremos diseñar una base de datos para un colegio. En la base de datos queremos tener toda la información sobre los alumnos y profesores del centro. Además queremos saber las asignaturas en las que están matriculados los alumnos y las asignaturas que imparte cada uno de los profesores. Además queremos llevar un historial con toda esta información de modo que sepamos las asignaturas que ha cursado cualquier alumno durante su periodo de formación y las calificaciones obtenidas. Del mismo modo, queremos tener almacenada la historia del profesor en el centro (asignaturas impartidas a lo largo de su trayectoria en el colegio).

Tenemos que almacenar la siguiente información para cada uno de los alumnos:

- DNI
- Nombre
- primer y segundo apellido del alumno.
- domicilio: dentro del domicilio se debe especificar la calle y número, así como la población, y la provincia donde reside.
- su fecha de nacimiento.

Para cada uno de los profesores, tenemos la siguiente información:

- DNI del profesor.
- su nombre y dos apellidos.
- su domicilio, con la misma información que los alumnos.
- su fecha de nacimiento.
- título (magisterio, físico, matemático, biólogo...)

Cada uno de los profesores imparte una serie de asignaturas. Habrá profesores que impartan una única asignatura y otros impartirán más de una. Hay que tener en cuenta que una asignatura, por ejemplo "Matemáticas", se puede impartir en varios cursos y también pueden ser diferentes profesores los que den Matemáticas en los diferentes cursos. Por tanto, esta situación hay que tenerla en cuenta en el diseño.

¿Cómo empezamos?

Pues bien, interpretemos la información que tenemos.

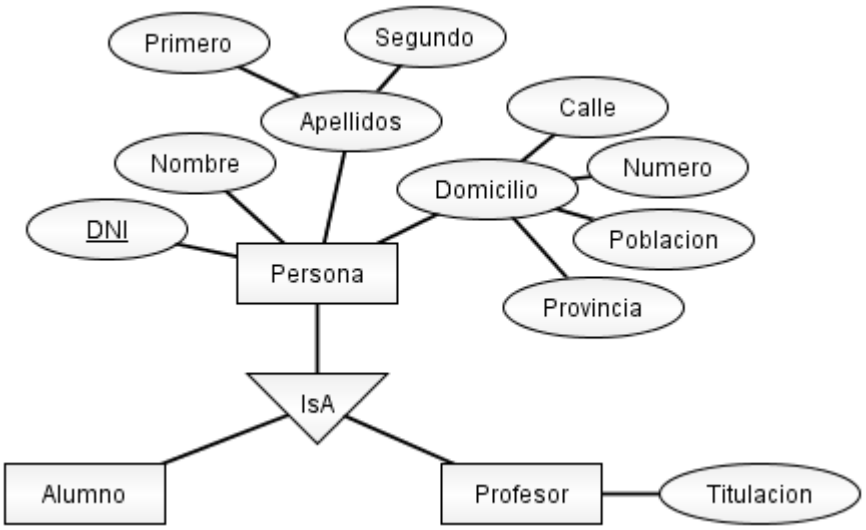
Si nos fijamos un poco en las características que se quieren tener de alumnos y profesores, no se diferencian mucho. Vemos que en lo único que se diferencian, como es obvio, es en la titulación. Además, no es buena idea guardar en el alumno las asignaturas matriculadas ni las asignaturas cursadas, porque no se sabe a priori cuáles ni cuántas son. Lo mismo ocurre para los profesores. Lo lógico es relacionar tanto alumnos como a profesores con otra entidad que llamaremos asignatura.

Una buena idea es definir una entidad padre con todos los atributos comunes y luego especializar cada uno de los hijos; crearemos una entidad Persona con todos los atributos

comunes de Alumno y Profesor y después pondremos los atributos específicos de cada uno de los hijos.

Abrimos DBDT, creamos un proyecto nuevo (creamos una nueva carpeta y la seleccionamos como directorio de trabajo) y representamos esta situación.

Importante: para cada uno de los atributos se debe especificar el dominio. Dicho dominio será el que se use a la hora de generar el script SQL.

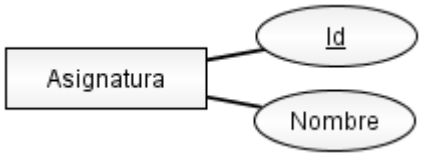


Vemos que efectivamente a la hora de representar los alumnos y los profesores en lo único que se diferencian es el la titulación de los profesores.

También es importante resaltar el hecho de que se ha marcado el atributo DNI como clave primaria: este atributo será el que distinga unívocamente dos alumnos o dos profesores.

¿Cómo representamos las asignaturas?

Una asignatura la distinguiremos de otra por un identificador (con esta opción conseguiremos resolver el problema anteriormente mencionado). De acuerdo a esto, una posible forma de definir la entidad asignatura sería la siguiente:

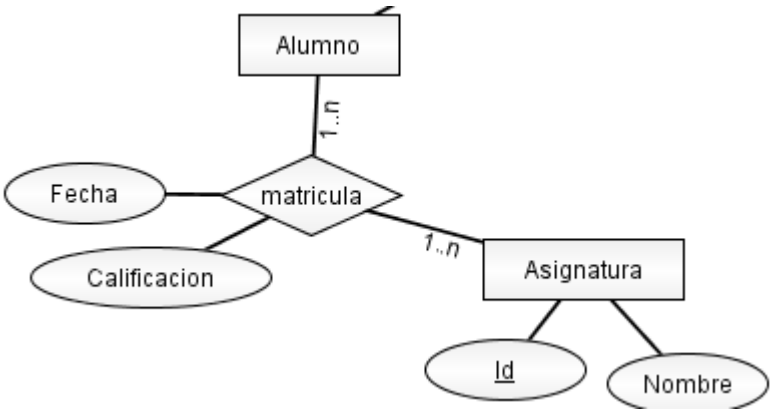


En una asignatura solamente tenemos que representar su nombre y un identificador que distinga una asignatura de otra. Con esta representación conseguimos lo que pretendemos.

¿Cómo representamos las asignaturas en las que ha estado y está matriculado un alumno?

Esta situación la representamos con una relación matricula. En esta relación relacionaremos (valga la redundancia) un alumno con las asignaturas. Deducimos que un alumno puede estar matriculado en muchas asignaturas y que para cada una de estas matriculas tenemos que guardar la calificación obtenida.

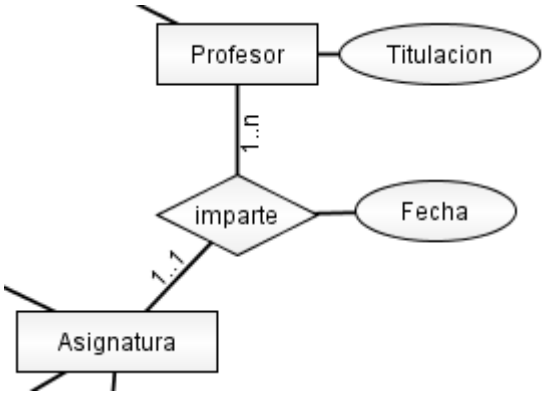
Una posible representación de esto puede ser la siguiente:



En la relación matrícula está involucrado el alumno y la asignatura. Se generará una tabla en la que estará el identificador del alumno (el DNI) y el identificador de la asignatura, además de la Fecha y de la Calificación obtenida.

¿Cómo representamos las asignaturas que imparte y ha impartido un profesor?

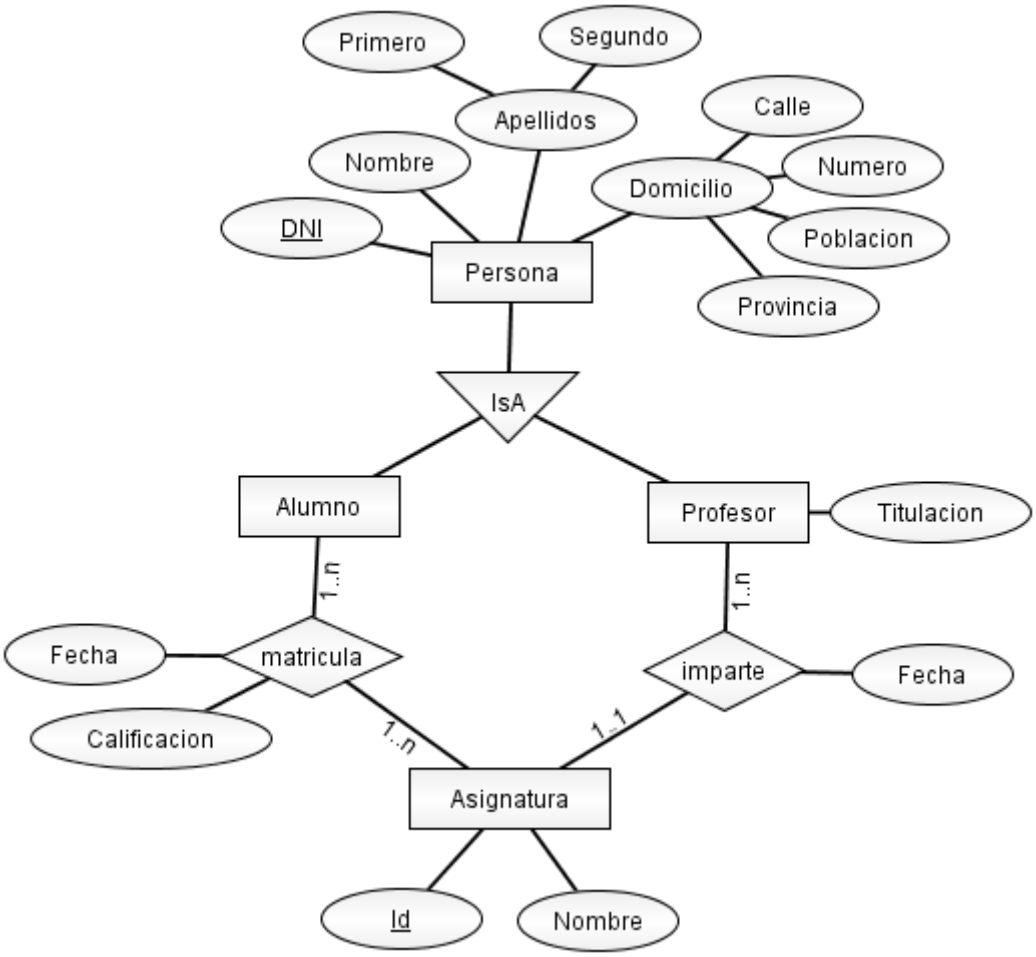
Procediendo de manera análoga a la anterior, podemos realizar el siguiente diseño:



Cada uno de los profesores impartirá una serie de asignaturas (de 1 a n), mientras que una asignatura solamente será impartida por un único profesor (de 1 a 1). Además en la relación guardamos la información sobre la fecha en la el profesor impartió la asignatura.

Bien. Llegados a este punto ya tenemos diseñada nuestra base de datos para el colegio.

El diagrama E/R final sería el siguiente:



Usamos las herramientas disponibles en DBDT para validar el diseño realizado y el sistema nos informa que el diseño es correcto.

Validando *imparte*

La relación imparte es de tipo Normal

ÉXITO: La relación posee las entidades adecuadas.

VALIDACIÓN REALIZADA CON ÉXITO

Ahora podemos utilizar las otras opciones disponibles para generar el script SQL correspondiente al diseño y exportarlo a un fichero .sql.

Posteriormente cargamos el fichero generado en un Sistema de Gestión de Bases de Datos y ya tenemos la base de datos creada.

Conclusiones y trabajo futuro.

Equipo de desarrollo.

Durante el desarrollo de este proyecto el equipo de diseño ha conseguido cumplir los objetivos que se había propuesto inicialmente.

Hemos conseguido una aplicación cercana al usuario, fácil de usar, con un posible uso docente muy útil, y sobre todo, que permite diseñar correctamente Bases de Datos Relacionales.

Las principales dificultades encontradas en el proceso de desarrollo han estado relacionadas con el objetivo de conseguir una interfaz fiel a los diseños en papel y que resultase cómoda de manejar. Gracias a una ardua labor, que ha servido tanto para resolver este escollo como para crecer como diseñadores/desarrolladores, se ha logrado obtener el aspecto que deseábamos para la aplicación.

Con respecto a los conocimientos que hemos requerido para la implementación de nuestra herramienta (tanto a nivel teórico como práctico) nos hemos bastado con los conocimientos adquiridos a lo largo de nuestra formación académica

Cierto es que la aplicación acepta diversas ampliaciones y mejoras, tanto de estructura como de contenido o de cualquier otro tipo. Por este motivo, el diseño empleado ha sido elegido para facilitar cualquier intento de mejora o actualización, en el caso de que una persona o grupo de personas intenten mejorar las cualidades de la herramienta desarrollada.

Profesor director.

El objetivo de este trabajo es el desarrollo (con fines didácticos y de apoyo a la docencia) de una aplicación que permite diseñar de forma sencilla una base de datos. Dicha aplicación proporciona al usuario las herramientas necesarias para crear el modelo conceptual de datos de alto nivel (modelo ER) y posteriormente, pulsando un solo botón, se valida dicho modelo y se genera un script de cliente DDL, liberando al usuario de esa tediosa tarea. Dicho script puede ser ejecutado posteriormente en algún SGBD comercial, como Oracle o MySQL.

Cabe destacar:

- Esta aplicación ha sido desarrollada con fines docentes y permitirá, a profesor y alumno, diseñar rápida y fácilmente los diagramas ER para cualquier base de datos.
- Se utiliza la notación de diagramas ER propuesta en una de las referencias incluidas en la bibliografía básica de la asignatura de "Bases de Datos y Sistemas de la Información", permitiendo al alumno hacer sus diseños rápidamente, sin necesidad de estudiar largos manuales, como ocurre con otras herramientas con la misma finalidad.
- Permite la creación de una base de datos tanto en Oracle como en MySQL sin más que ejecutar un script.
- Se incluye manual de usuario, tanto en la memoria del proyecto como en la propia aplicación.
- La aplicación se ha desarrollado de forma que se pueda ejecutar tanto en entornos Windows como Unix.
- Permite la ampliación de funcionalidad de forma sencilla.

Referencias.

Bibliografía.

Fundamentos de Bases de Datos.
Silberschaz, Koth y Sudarshan.
4ª Edición, Editorial McGrawHill.

J2EE Design Patterns.
Jonathan Kaplan.
1ª Edición. Editorial O'Reilly & Associates

Web.

[Std_830]

Estándar IEEE Std. 830-1998 para la realización de la SRS.

http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html

MySQL con Clase

Web con información sobre Bases de Datos

http://www.mysql.conclase.net

Java Sun

http://java.sun.com